Modélisation au niveau transactionnel de l'architecture et du contrôle relatifs à la gestion d'énergie de systèmes sur puce

Hend Affes

UNIVERSITE NICE SOPHIA ANTIPOLIS

ECOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE

Pour l'obtention du grade de

Docteur en Sciences

de l'Université Nice Sophia Antipolis

Mention : Informatique

Présentée par

Hend Affes

Modélisation au niveau transactionnel de l'architecture et du contrôle relatifs à la gestion d'énergie de systèmes sur puce

Directeur de thèse : **M. Michel Auguin**

Soutenue le 18 Décembre 2015

JURY

M. Frédéric Rousseau	Professeur, Univ.Grenoble.Alpes	Rapporteur
M. Eric Senn	Maître de conférence, UBS	Rapporteur
Mme. Patricia Guitton-ouhamou	Ingénieur, Intel	Examinatrice
M. François Verdier	Professeur, Université de Nice	Examinateur
M. François Pêcheux	Professeur, UPMC - LIP6	Examinateur
M. Michel Auguin	Directeur de recherche, CNRS	Directeur de thèse

À mes parents,

À mes frères,

À ma sœur,

À mon mari,

À tous les membres de ma famille sans aucune exception.

Et à tous ceux que ma réussite leur tient à cœur.

Je dédie ce travail.

Remerciement

Au terme de ce travail, je tiens à exprimer mes remerciements envers toutes les personnes qui ont contribuées de près ou de loin à l'achèvement de cette précieuse expérience.

Je témoigne ma profonde reconnaissance à mon encadreur de thèse Monsieur Michel Auguin, pour m'avoir dirigée, guidée, aidée, soutenue et encouragée tout au long de l'élaboration de ce travail. Je le remercie pour toute la patience et la disponibilité dont il a fait preuve à mon égard. J'apprécie énormément ses hautes qualités scientifiques et ses valeurs humaines exemplaires. Merci pour tout ! Les mots ne peuvent pas exprimer toute ma gratitude.

Mes remerciements s'adressent aux membres de jury : Monsieur Frédéric Rousseau et Monsieur Eric Senn pour avoir accepté d'être rapporteurs de ce travail. Je tiens à remercier Monsieur François Pêcheux, Monsieur François Verdier et Madame Patricia Guitton-Ouhamou pour avoir accepté d'être examinateurs de ce travail.

Je remercie tous les membres du laboratoire LEAT, dans lequel j'ai passé mes trois ans de thèse. Merci aussi à tous mes collègues du laboratoire qui se reconnaitront ici pour leur amitié.

Un grand merci à mes parents : Mohamed et Bahija. Aucun hommage ne pourrait être à la hauteur de l'amour dont ils ne cessent de me combler. Vos conseils ont toujours guidé mes pas vers la réussite. Certainement, tout ce que j'ai pu réaliser est grâce à vous. Que Dieu vous procure bonne santé et longue vie.

A mes chers frères Faiez et Mohamed et à ma chère sœur Marwa, merci pour vos encoura-

gements et vos soutiens. Vous êtes toujours dans mes pensées et dans mon cœur.

A mon cher mari Mohamed, qui a su me soutenir avec plein d'amour et délicatesse pendant les moments difficiles, merci pour ta présence, ta compréhension, ta patience, ton aide, ton soutien, ton amour... Merci beaucoup !

Abbréviation

AT	**A**pproximately **T**imed
CA	**C**ycle **A**ccurate
CPF	**C**ommon **P**ower **F**ormat
CD	**C**lock **D**omain
ClkARCH	**Clk ARCH**itecture
ClkST	**Clk S**tate **T**able
ClkCTr	**Clk C**control **Tr**ansaction
CM	**C**lock **M**anagement
DE	**D**esign **E**lement
DMAC	**D**irect **M**emory **A**ccess **C**ontroller
DPLL	**D**igital **P**hase **L**ocked **L**oop
DST	**D**PLL **S**tate **T**able
DPC	**D**omain **P**ower **C**ontroller
DPM	**D**ynamic **P**ower **M**anagement
DSP	**D**ynamic **P**ower **S**witching
DVFS	**D**ynamic **V**oltage and **F**requency **S**caling
ESL	**E**lectronic **S**ystem and **L**evel
GC	**G**enerated **C**lock
HDL	**H**ardware **D**escription **L**anguage
TCL	**T**ool **C**ommand **L**anguage
IP	**I**ntellectual **P**roperty
LT	**L**oosely **T**imed
OS	**0**perating **S**ystem
PD	**P**ower **D**omain

OPP	**O**perating **P**erformance **P**oint
PST	**P**ower **S**tate **T**able
PSM	**P**ower **S**tate **M**achine
PSL	**P**roperty **S**pecification **L**anguage
PSw	**P**ower **Sw**itch
PM	**P**ower **M**anager
PRM	**P**ower and **R**eset **M**anagement
PMP	**P**ower **M**anagement **P**oint
PMU	**P**ower **M**anagement **U**nit
PRCM	**P**ower **R**eset, and **C**lock **M**anager
PV	**P**rogrammer **V**iew
PVT	**P**rogrammer **V**iew with **Timing**
PwARCH	**Pw**er **ARCH**itecture
QoS	**Q**uality **o**f **S**ervice
RCG	**R**eset **C**lock **G**enerator
RCT	**R**equest **C**apable **T**arget
RTL	**R**egister **T**ransfer **L**evel
TL	**T**ransaction **L**evel
TLM	**T**ransaction **L**evel of **M**odeling
TGI	**T**ight **G**enerator **I**nterface
UPF	**U**nified **P**ower **F**ormat
VP	**V**irtual **P**latform

Résumé :

Les systèmes embarqués sur puce (SoC) envahissent notre vie quotidienne. Avec les progrès de la technologie, ils intègrent de plus en plus de fonctionnalités complexes engendrant des besoins accrus en temps de calcul et taille de mémoire. Alors que la complexité de ces systèmes est une tendance clé, la consommation d'énergie est aussi devenue un facteur critique pour la conception de SoC.

Dans ce contexte, nous avons étudié une approche de modélisation au niveau transactionnel qui associe à un modèle fonctionnel SystemC-TLM une description d'une structure de gestion d'un arbre d'horloge décrit au même niveau d'abstraction. Cette structure développée dans une approche de séparation des préoccupations fournit à la fois l'interface pour la gestion de puissance des composants matériels et pour le logiciel applicatif. L'ensemble des modèles développés est rassemblé dans une librairie *ClkARCH*. Pour appliquer à un modèle fonctionnel un modèle d'un arbre d'horloge, nous proposons une méthodologie en trois étapes : la spécification, la modélisation et la simulation. Une étape de vérification en simulation est aussi considérée basée sur des contrats génériques de type assertion.

De plus, nos travaux visent à être compatibles avec des outils de conception actuels. Nous avons proposé une représentation d'une structure de gestion d'horloge et de puissance dans le standard IP-XACT permettant de produire les descriptions C++ des structures de gestion de puissance du SoC.

Par ailleurs, nous avons proposé une approche de gestion de puissance basée sur l'observation globale des états fonctionnels du système dans le but d'éviter ainsi des prises de décisions locales peu efficaces à une optimisation de l'énergie.

Mots Clé : Systèmes sur Puce, Niveau Transactionnel, Gestion et Vérification de l'arbre d'horloge *Clock Domain*, *clock intent*, *power intent*, Assertions, Prédiction de mode de puissance.

Abstract :

Embedded systems-on-chip (SoC) invade our daily life. With advances in semiconductor technology, these systems integrate more and more complex and energy-intensive features which generate increasing computation load and memory size requirements. While the complexity of these systems is a key trend, the energy consumption has also become a critical factor for SoC design.

In this context, we have studied a modeling transactional level approach allowing a description of a clock tree and its management structure to be associated with a functional model, both described at the same abstraction level. This structure developed in a separation of concerns approach provides both the interface to the power consumption management of the hardware components and the application software. All the models developed to describe such a structure are gathered in a ClkArch library. To apply to a SystemC-TLM functional model a clock tree intent with its control part, we propose a methodology based on three steps : specification, modeling and simulation. A verification step based on simulation is also considered using contracts of assertion type.

This work aims to build a modelling approach on current design tools. So we propose a representation of a clock and power management structure in the IP-XACT standard allowing a C++ description of the SoC power management structures to be generated.

Finally, a power management strategy based on the global functional states of the components of the system architecture is proposed. This strategy avoids local decision-making unsuited to optimized overall power/energy management.

Keywords : Systems-on-Chip, Transaction Level Modeling, Clock-Management Design and Verification, Clock Domain, clock intent, power intent, assertion-based contracts, power state prediction.

Table des matières

Table des figures

List of Algorithms

Chapitre I

Introduction générale

1 Préambule

Les systèmes embarqués sur puce (*System on Chip - SoC*) envahissent notre vie quotidienne et professionnelle. En effet, ces systèmes couvrent une partie très large des innovations réalisées dans divers domaines comme la robotique, l'aviation, les télécommunications, l'automobile, la sécurité, la médecine, etc. Avec les progrès de la technologie toujours d'actualité, ces systèmes intègrent de plus en plus de fonctionnalités complexes et gourmandes en énergie engendrant des besoins accrus en termes de temps de calcul et de taille de mémoire consommée. Par exemple, les

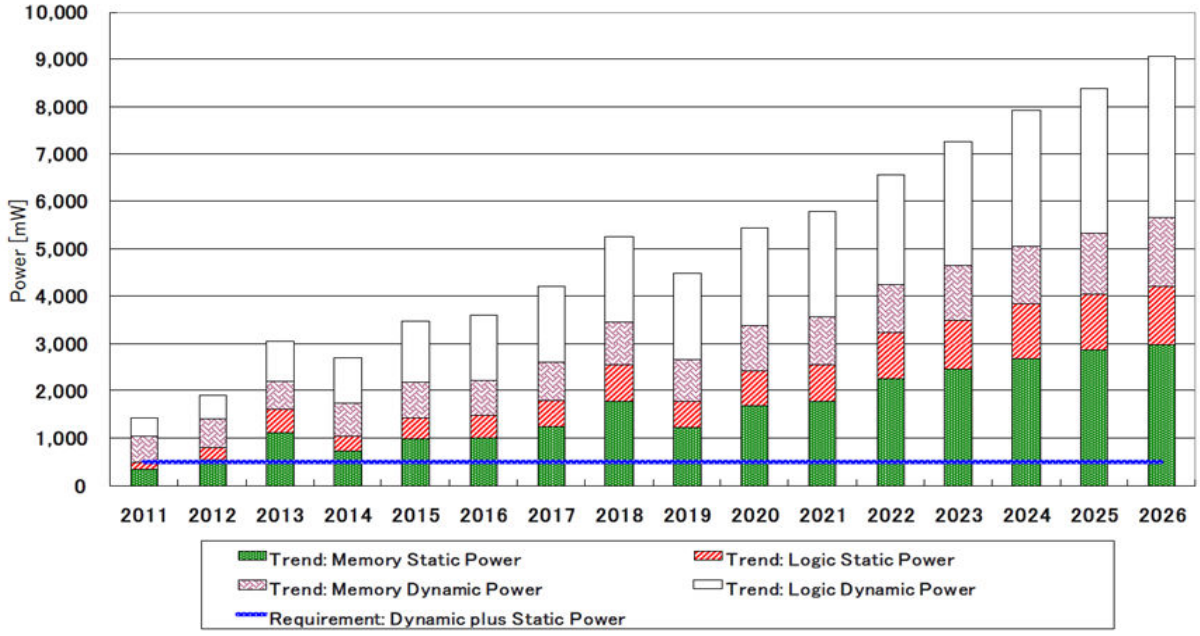


Figure I.1 – Tendance de la consommation d'énergie des SoC selon l'IRTS (Source : Silicon Integration Initiative (Si2), dérivé de l'ITRS 2011)

téléphones intelligents d'aujourd'hui, hormis leur fonction première qui est de téléphoner, sont devenus suffisamment sophistiqués pour servir aussi de GPS, d'appareil photo, de télévision, de lecteur de musique, de console de jeux, etc. Alors que la complexité de conception est une tendance clé, la consommation d'énergie est aussi un facteur critique pour la conception de SoC.

La Figure I.1 montre l'évolution de la consommation par rapport aux différentes sources d'énergie pour les systèmes sur puce modernes. Cette Figure illustre également l'augmentation de puissance dissipée observée ou prédite année après année dans chacune des parties du système. Ainsi, que ce soit la composante statique de la puissance due aux courants de fuites ou la composante dynamique liée aux changements d'états logiques du système, la tendance va dans le sens de leur augmentation continue.

Cette augmentation de puissance dissipée est principalement due au fait que la tension d'alimentation des transistors ne diminue plus dans les mêmes proportions que la réduction de la taille des transistors entre deux nœuds technologiques. Ceci est illustré dans le tableau suivant extrait d'une analyse effectuée par l'ITRS en 2010.

Nœud Technologique	Surface silicium	Capacité	Tension d'alimentation	Fréquence Max	Densité de puissance
45 -> 32 nm	x 0,57	x 0,66	x 0,925	x 1,1	x 1,09
32 -> 22 nm	x 0,57	x 0,66	x 0,950	x 1,08	x 1,13
22 -> 14 nm	x 0,57	x 0,66	x 0,975	x 1,05	x 1,16
14 -> 10 nm	x 0,57	x 0,66	x 0,985	x 1,04	x 1,17

Tableau I.1 – Analyse d'évolution de caractéristiques technologiques effectuée par l'ITRS en 2010

Ainsi, nous observons que la surface de silicium est bien réduite d'un facteur constant entre deux nœuds technologiques (la Loi de *Moore*), ce qui conduit à une capacité de commutation réduite dans des proportions identiques (ce qui agit favorablement sur la puissance dynamique). Cependant, la tension d'alimentation ne suit pas du tout la même évolution. Ceci est principalement dû au fait que réduire trop cette tension a pour effet d'augmenter les courants de fuite (et donc la puissance statique dissipée) et rend le circuit de plus en plus sensible aux bruits

induits par les changements d'états (*noise margin*) [HH08], au risque d'obtenir des fonctionnements erronés. La tension d'alimentation ne diminuant ainsi que dans de faibles proportions, la puissance dynamique reste alors élevée et concentrée dans une surface plus réduite avec pour conséquence une augmentation de la densité de puissance. Cette densité de puissance ayant un effet immédiat sur la quantité de chaleur produite par unité de surface, nous atteignons rapidement une température limite qui entraîne soit un vieillissement prématuré du circuit soit même sa destruction par effet combiné entre température et puissance statique. Pour limiter cette température, nous sommes obligés d'agir sur l'autre paramètre qui impacte la consommation : la fréquence d'horloge. Ce phénomène est ainsi mis en évidence sur la Figure I.2 où depuis 2005 nous assistons à une augmentation sur un même rythme du nombre de transistors intégrés qui dans le même temps s'accompagne d'une puissance dissipée qui tend à se stabiliser avec pour conséquence une fréquence qui n'évolue plus, un nombre de cœurs par puce qui ne croît plus aussi rapidement et donc une performance globale qui fléchit également.

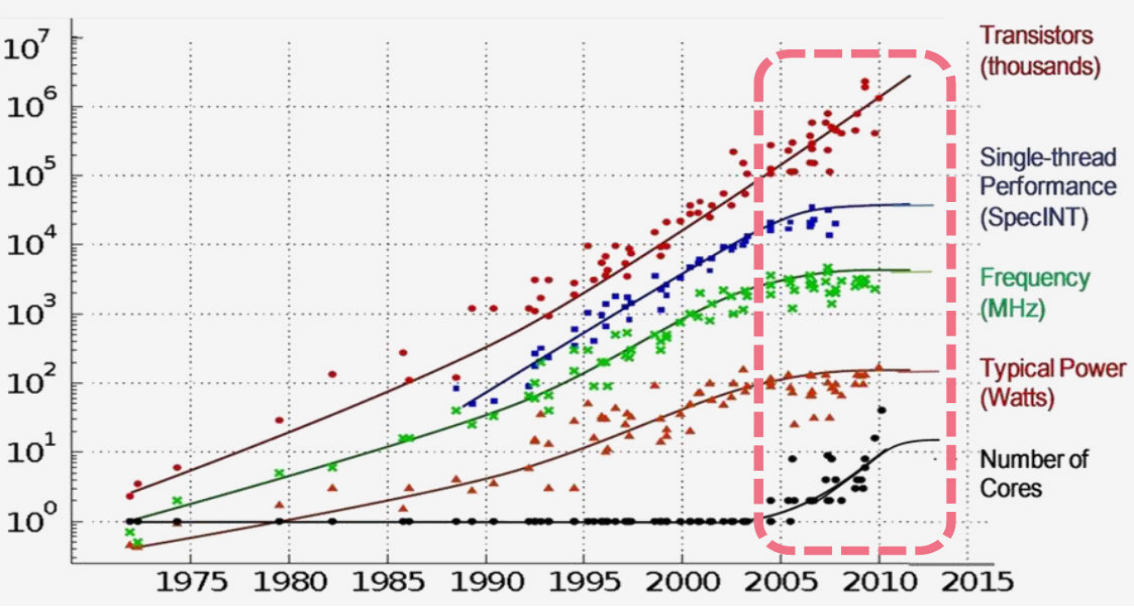


Figure I.2 – Évolutions des caractéristiques principales des circuits complexes sur 40 ans [Car15]

Si ce caractère thermique est la cause de ce ralentissement dans l'exploitation des capacités d'intégration des technologies avancées, son origine se trouve bien dans la densité de puissance dissipée par le circuit lorsque celui-ci est sous tension et de plus en fonctionnement. Ainsi, depuis de nombreuses années, différentes techniques ont été développées dans les circuits pour tenter de réduire la puissance dissipée [TR11]. Ces techniques ont été aussi largement la conséquence

d'une volonté de prolonger le plus possible la durée de fonctionnement des équipements mobiles alimentés sur batterie (les téléphones portables par exemple) ou sur pile (comme certaines montres connectées à pile). En conséquence, la conception et la vérification de circuits basse consommation est en soit une problématique qui a fait l'objet de très nombreux travaux. Ceci illustre les difficultés soulevées pour obtenir des solutions efficaces en énergie et conformes aux objectifs souhaités de performance fonctionnelle.

Depuis peu d'années est apparu le terme de *Dark Silicon* [SGM+14]. Ce terme fait référence au fait qu'il n'est plus possible d'utiliser simultanément l'ensemble des fonctionnalités d'un circuit à sa fréquence d'horloge typique sans que cela n'entraîne un dépassement de la quantité de puissance dissipée autorisée qui conduit à la température maximum permise. En se référant à ce mode d'utilisation d'un circuit, cela ajoute une dimension supplémentaire par rapport aux précédentes techniques de gestion de puissance. En effet, il s'agit alors de gérer une répartition spatiale des activités sur le circuit en coordination avec les techniques classiques de gestion de puissance dissipée. Ce type de mécanisme voit déjà le jour avec par exemple l'approche initiée par la société ARM avec le système multi-cœurs *Big.LITTLE* [Goo12] qui intègre une architecture hétérogène avec une vue programmeur homogène. La gestion d'énergie dans ce type d'architecture doit tenir compte de l'hétérogénéité de l'architecture pour obtenir les meilleures optimisations en énergie en fournissant par ailleurs un niveau de performance conforme aux attentes.

En parallèle à ces évolutions technologiques et architecturales, le besoin croissant d'optimisation de la puissance consommée a conduit à agir à tous les niveaux du flot de conception et de développement d'un circuit afin d'y insérer les mécanismes adaptés à cette optimisation. Ainsi, dans un premier temps les efforts étaient plutôt concentrés sur les niveaux logiques et RTL mais ils se sont rapidement étendus au niveau du système complet également. Ceci est illustré sur la Figure I.3 qui montre que les niveaux ESL comportemental et architecture concentrent aujourd'hui le principal effort d'optimisation. Pourtant, si des outils matures existent aux niveaux RTL et en dessous, avec des normes mises en place correspondant à ce domaine, les outils commerciaux sont largement moins diffusés et utilisés au niveau ESL pour traiter ce problème d'optimisation de puissance au niveau système.

Cette thèse se positionne ainsi au niveau ESL pour contribuer à proposer une approche par simulation pour aider à la conception et à l'analyse d'architectures de systèmes sur puces optimisées en consommation de puissance.

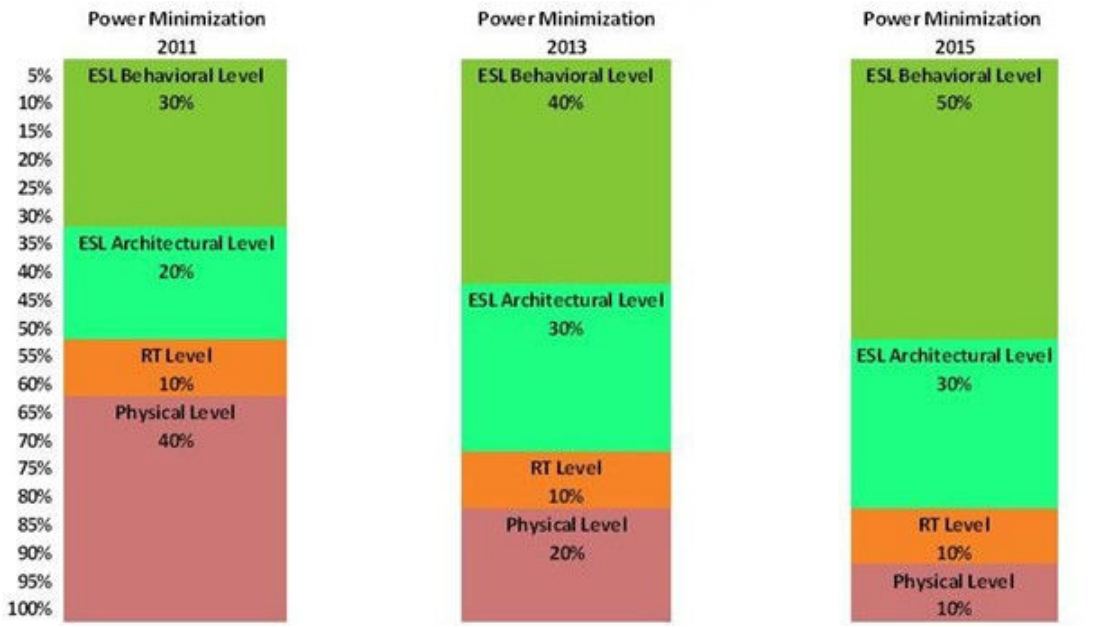


Figure I.3 – Évolutions de l'optimisation de puissance dans les différentes phases de la conception [TR11]

2 Contributions

Le sujet de cette thèse est d'étudier les concepts et les mécanismes à considérer dans un modèle fonctionnel au niveau transactionnel pour la réduction de la consommation d'énergie. L'étude réalisée nous a permis de formuler un ensemble de besoins et de défis concernant la gestion de la consommation d'énergie au niveau transactionnel.

Nous avons comme premier objectif de fournir une méthodologie d'insertion de stratégies de gestion d'horloge au niveau transactionnel. L'intérêt principal de la méthodologie proposée concerne sa capacité de spécification et de gestion d'une infrastructure à faible consommation conduisant à des modèles fonctionnels TLM bien structurés. De plus, elle vise à rendre possible l'exploration des différentes architectures à faible consommation d'énergie avec différentes décompositions en *Clock Domains* afin d'évaluer les effets d'une gestion d'énergie sur la performance d'un système et sa fonctionnalité. Cette méthodologie respecte le principe de la séparation entre les aspects fonctionnels et ceux orientés gestion d'horloge.

Structurer manuellement des architectures selon des stratégies de gestion de puissance nécessite un effort important de modélisation et peut conduire à des erreurs de conception peu triviales à identifier. En effet, ces erreurs peuvent résulter de comportements incohérents entre ceux induits par les décisions issues de la gestion de puissance et ceux relatifs aux états fonctionnels produits par l'exécution du modèle SystemC-TLM. Dans ce contexte, notre deuxième

objectif est de fournir une approche structurelle en utilisant le standard IP-XACT [IP-] pour produire des modèles de simulation SystemC-TLM d'architectures matérielles intégrant la description des structures de gestion de puissance basées sur la décomposition en *Clock Domains* et *Power Domains*.

Enfin, nous avons fixé comme troisième objectif, la définition d'une stratégie dynamique de gestion de puissance pour les systèmes sur puce. Cette thèse contribue à la proposition d'une stratégie qui se base sur l'observation et l'analyse des états fonctionnels globaux des blocs IP du système. Cette stratégie est indépendante du système d'exploitation ou de l'application tandis qu'elle peut être contrôlée facilement par ces couches logicielles par l'intermédiaire des contraintes sur les modes de puissance qui doivent respecter cette stratégie.

3 Plan du mémoire

Ce manuscrit présente les travaux réalisés au cours des trois années de thèse. Il est structuré en six chapitres incluant la présente introduction et une conclusion générale.

Le chapitre II présente, dans un premier temps, une introduction du contexte général de nos travaux : un rappel sur le flot de conception des systèmes sur puces, les techniques de gestion d'énergie ainsi que les standards utilisés dans les approches de conception à faible consommation. Dans un second temps, il dresse un état de l'art des techniques de gestion et de modélisation d'énergie principalement au niveau transactionnel.

Dans le chapitre III, nous présentons d'abord les objectifs de notre travail ainsi que les concepts et les caractéristiques clés pour la modélisation de la distribution d'horloge au niveau TLM des systèmes sur puce à faible consommation. Puis, nous détaillons le flot de la méthodologie proposée en expliquant les principales caractéristiques des modèles développés, supports à cette méthodologie. Nous montrons comment nous avons appliqué notre méthodologie sur un exemple de modèle d'une plateforme SystemC-TLM. La dernière partie du chapitre est une discussion à la lumière des résultats obtenus des apports et des limitations de l'approche développée et des améliorations qui peuvent être apportées.

Le chapitre IV présente une approche de modélisation structurelle des architectures fonctionnelles augmentées par une stratégie de gestion de puissance au niveau transactionnel. Cette approche est basée sur le flot du standard IP-XACT. Après la description de notre approche, nous décrivons les générateurs de code produisant les descriptions C++ des structures de gestion de puissance du SoC.

Dans le chapitre V, nous décrivons notre stratégie de gestion de puissance pour les systèmes

sur puce à faible consommation basée sur l'analyse des états fonctionnels. Nous détaillons le flot de notre stratégie. Puis, nous présentons les résultats de simulation obtenus sur différents cas d'étude.

Nous concluons cette thèse dans le dernier chapitre et nous identifions des perspectives de futurs travaux de recherche.

Dans cette thèse, nous utilisons assez largement des termes techniques anglophones dans la rédaction pour désigner différents objets dont la traduction française n'est pas concise ou évocatrice.

Chapitre II

Contexte et état de l'art

1 Introduction

Les contraintes que doit prendre en compte la conception de systèmes sur puce deviennent de plus en plus nombreuses. En effet, elles sont la conséquence paradoxale de la capacité physique des puces à intégrer un nombre toujours croissant de portes logiques ce qui permet a priori de satisfaire les divers besoins des nouveaux produits. L'évolution du nombre de transistors dans une puce a ainsi suivi la loi de *Moore* avec une croissance d'environ 100% tous les deux ans pendant des décennies. Cette évolution devrait se poursuivre pendant au moins les dix prochaines années à un rythme éventuellement moins rapide. Dans ce contexte où la réduction de la tension d'alimentation n'a pas suivi la même pente, la gestion de la consommation d'énergie et des aspects thermiques associés est devenue une préoccupation majeure des concepteurs de SoC puisqu'elle risque de provoquer des goulots d'étranglement au niveau des ressources (une ressource qui voit sa fréquence de fonctionnement réduite exécute moins d'opérations par seconde) et ainsi ralentir les performances des solutions définies.

Ce chapitre est consacré à la présentation des divers concepts et notions de base nécessaires à la compréhension de nos travaux de recherche et à les situer par rapport aux travaux publiés dans le domaine. D'abord, nous introduisons l'évolution du flot de conception des SoC. Nous décrivons, ensuite, les techniques et les standards de conception à faible consommation qui concernent la gestion de la réduction de la puissance des systèmes sur puce. Puis, nous présentons les travaux principaux dans le domaine de la modélisation et de l'analyse de puissance au niveau système (*Electronic System Level* - ESL).

2 Contexte

2.1 Le flot de conception d'un système sur puce (*System-on-Chip* - SoC)

Un flot de conception de SoC intègre généralement de multiples niveaux d'abstraction, de la spécification initiale jusqu'au produit final. Ces niveaux offrent différents compromis entre la vitesse d'analyse et de synthèse et la précision des résultats obtenus. Au fur et à mesure que la modélisation passe d'un niveau élevé à un autre plus bas, les détails des solutions architecturales augmentent progressivement et l'espace de ces solutions devient de plus en plus réduit. La Figure II.1 résume les niveaux d'abstraction durant un flot de conception de SoC. Idéalement, le flot de conception commence par le niveau le plus haut, qui est classé le niveau le plus abstrait et le moins précis, et se raffine soit automatiquement, soit manuellement vers le niveau le plus bas, qui est inversement classé le plus concret et le plus précis.

La conception des SoC au niveau RTL (*Register Tranfert Level*) et les niveaux au-dessous, qui sont des niveaux précis au cycle et bit près, se heurte au problème du temps nécessaire à la production et à la vérification du système complet du fait de la complexité liée au nombre très élevé d'états du système qu'il s'agit de représenter et de gérer. Ces niveaux sont donc incompatibles avec un objectif d'exploration de solutions optimisées en performance et consommation d'énergie. Pour y remédier, les concepteurs s'intéressent de plus en plus du niveau système (ESL) qui présente un niveau de conception plus abstrait. Avec cette abstraction, la conception de SoC s'effectue plus rapidement et plus facilement et peut dans certains cas même s'effectuer sans perdre de précision [VPB06]. Cela permet à l'équipe de conception de simuler et valider rapidement les spécifications initiales ainsi que d'évaluer un grand nombre d'alternatives de mises en œuvre.

Dans ce travail, nous nous focalisons sur le niveau système (ESL) qui regroupe les différents niveaux d'abstraction au-dessus du niveau RTL représentés dans la Figure II.1. Dans la suite, nous décrivons brièvement les principales caractéristiques de chaque niveau dans le flot de conception de SoC.

2.1.1 Niveau Algorithme : (*Algorithmic Level* - AL)

Le niveau dit *Algorithme* est le niveau le plus haut (au-dessous de la spécification) d'un flot de conception. A ce niveau, l'application est décrite sous forme algorithmique basée sur une spécification standard ou une documentation existante. L'architecture matérielle du système n'est pas définie et les modèles algorithmiques sont décrits en utilisant un langage de description

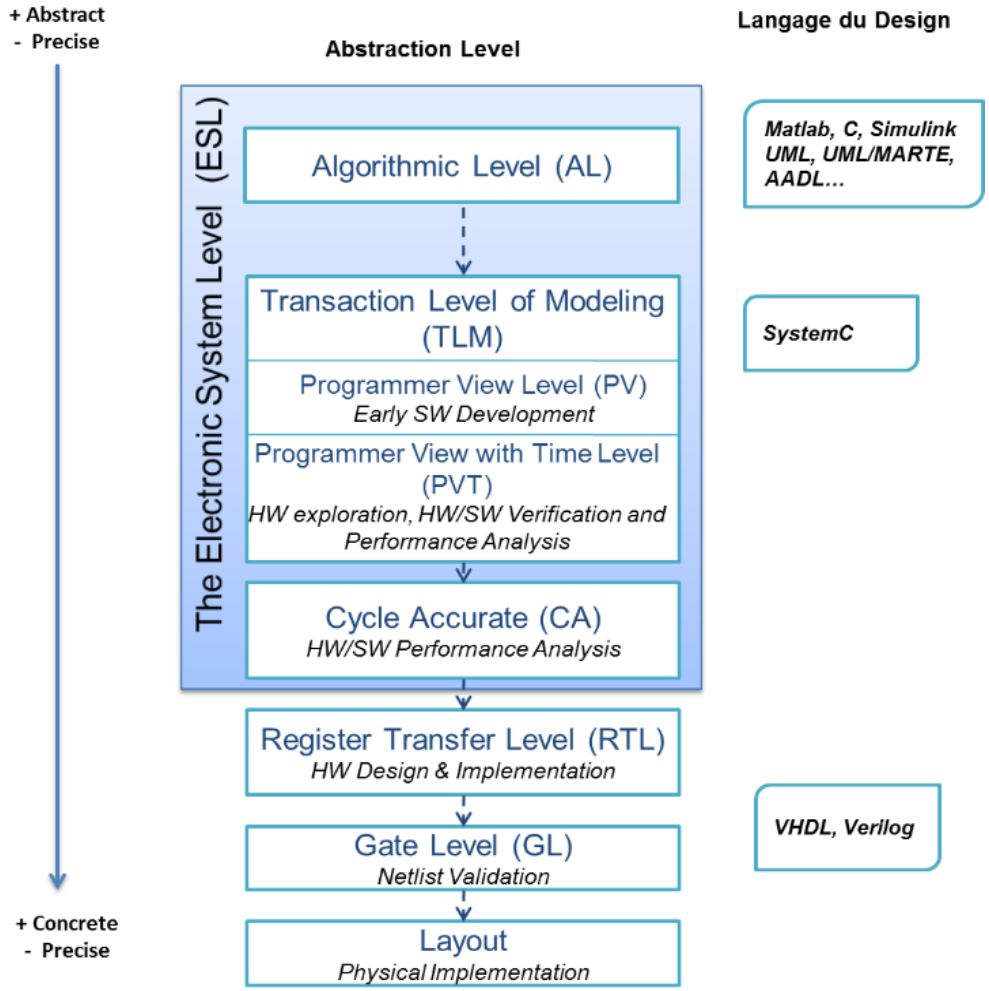


Figure II.1 – Niveaux d'abstraction dans le flot de conception d'un SOC [Ati08]

haut niveau tels que Matlab ou C++. Ces modèles permettent de réaliser une vérification fonctionnelle précoce de l'application, d'évaluer les complexités réciproques des fonctionnalités représentées et ce afin de les répartir ensuite efficacement en tâches matérielles et logicielles.

2.1.2 Niveau transactionnel : (*Transaction Level Modeling* - TLM)

Le terme TLM dénote une approche de modélisation basée sur les transactions et fondée sur des langages de programmation de haut niveau comme SystemC [sys]. Le terme de transaction désigne les messages échangés entre les composants d'un système. Ceci traduit le fait que la communication s'effectue par l'échange de données suivant un protocole abstrait (et non précis au signal près). Depuis l'année 2000, plusieurs définitions et classifications des différents

niveaux TLM ont été présentées dans la littérature comme dans [Don04], [CG03], [FMPP04] et [GSGS02]. Toutes ces publications ont des points communs. Premièrement, les composants sont modélisés comme des modules avec un ensemble de processus simultanés. Les transactions entre ces modules sont simplifiées en utilisant des méthodes de communication via des canaux [ASJJM05]. Deuxièmement, les aspects de communication sont séparés des aspects de calculs.

Par ailleurs, TLM est présenté comme une taxonomie de plusieurs sous-niveaux. Dans notre travail, nous nous intéressons en particulier aux deux sous-niveaux décrits ci-dessous :

- *Programmer View* (PV) ou *Loosely-Timed* (LT)
- *Programmer View with Timing* (PVT) ou *ApproximatelyTimed* (AT)

2.1.2.1 *Programmer View* (PV) :

La précision des architectures des systèmes décrites au niveau TLM ne se traduit pas directement par une précision temporelle de simulation. En effet, les mécanismes de communication implémentés dans les interfaces des composants sont simplifiés sans prendre en compte le temps. Nous parlons dans ce cas des modèles PV. Ce sous-niveau d'abstraction est principalement utilisé pour valider au niveau algorithmique le système complet en exécutant l'application finale sur le modèle transactionnel d'architecture. Néanmoins, des propriétés extra-fonctionnelles comme le temps d'exécution et la consommation d'énergie, sont soit omises soit approximées. Certains détails architecturaux tels que l'activité des mémoires cache et l'arbitrage de bus ne sont pas modélisés au sous-niveau PV. Ces détails ont une grande influence sur la précision des modèles de simulation et les mesures de performance, mais les intégrer dans les modèles de simulation ralentit la simulation. Ce type de détails est plutôt modélisé au sous-niveau PVT.

2.1.2.2 *Programmer View with Timing* (PVT) :

Le sous-niveau PVT représente la même vue fonctionnelle que le sous-niveau PV mais en ajoutant des annotations temporelles plus précises. De plus, des interconnexions et des arbitrages de communication sont modélisés. Ainsi, le niveau PVT est plus précis que le niveau PV. Il permet d'extraire les solutions pertinentes dans un temps raisonnable tout en offrant un niveau de précision proche de celui obtenu par des niveaux d'abstraction plus raffinés.

2.1.3 Le niveau Cycle Précis : (*Cycle Accurate Level* - CAL)

Le niveau Cycle Précis fournit une description précise de l'état du modèle à chaque cycle d'horloge. Ce niveau d'abstraction représente un point médian entre la simulation traditionnelle

pilotée par des événements (les transitions entre états sont détaillées dans le cycle d'horloge) et des modèles de transaction de haut niveau (les états n'évoluent pas entre deux transactions successives sur le bus). Au niveau calcul, une description de la micro-architecture interne du processeur (pipeline, prédiction de branchement, cache, etc.) est réalisée, tandis qu'au niveau de la communication, un protocole de communication précis au bit près est adopté. Une telle précision des modèles de cette description améliore la qualité de l'estimation de performance et permet d'identifier des erreurs de comportement. Cependant, ces modèles présentent une vitesse limitée de simulation (mais toutefois généralement un ordre de grandeur plus rapide que le transfert niveau aux registres (RTL)) et nécessitent un effort de modélisation significatif.

2.1.4 Le niveau transfert de registres : (*Register Tranfert Level* - RTL)

Le niveau RTL se réfère au niveau d'abstraction auquel un circuit est décrit suivant des transferts synchrones entre les unités fonctionnelles, telles que les multiplieurs, les unités arithmétiques et logiques, et les fichiers de registres [GSGS02]. Ce niveau exprime l'ordonnancement au cycle d'horloge près des opérations et des transferts de données. Pour l'aspect structurel, ces opérations sont projetées sur des ressources matérielles élémentaires (registres, opérateurs arithmétiques, etc.). En effet, à ce niveau de description, les transferts de données et les opérations logiques entre les registres sont décrits via des signaux. Des langages de description matérielle (HDL) comme Verilog, VHDL et ESTEREL [Ger00] sont utilisés pour décrire des modèles RTL.

2.1.5 Le niveau porte logique : (Gate Level - GL)

Le niveau logique décrit le comportement sous forme d'équations booléennes, qui peuvent elles-mêmes se traduire sous la forme d'une interconnexion de portes logiques d'une part, et d'autre part de cellules élémentaires placées et routées d'une bibliothèque technologique de type ASIC (*Application-Specific Integrated Circuit*). Cette description élimine de nombreux détails propres au niveau physique (par exemples ceux liés au placement/routage) en se concentrant uniquement sur la description des portes logiques (ET, OU, etc.) et leurs connexions.

2.1.6 Le niveau dessin de masque : (*Layout Level*)

Le niveau dessin de masque modélise le comportement électrique sous forme d'équations différentielles ou de fonctions de transfert. Une représentation structurelle décompose le circuit en une interconnexion de transistors et une représentation géométrique qui détaille le placement et le routage de ces transistors sur le substrat de silicium. Ce niveau *Layout* est la description

la plus précise de la puce. À ce niveau, l'emplacement et la conception de chaque transistor sont précisément connus et doivent être soigneusement vérifiés. Enfin, les transistors et leurs connexions sont transposés sur des masques qui sont utilisés dans le processus de fabrication. Ces masques sont très coûteux à fabriquer : il est classique d'estimer qu'un jeu de masques en technologie 65 nm peut coûter jusqu'à 1 million de dollars. Par conséquent, les erreurs de conception dans le matériel peuvent s'avérer économiquement désastreuses à corriger en raison de la nécessité de reconstruire ces jeux de masques.

2.2 Les concepts de base du niveau transactionnel (TLM)

Nos travaux se positionnent au niveau ESL et plus précisément au niveau transactionnel. Pour cela, nous présentons ici une description des concepts introduits dans le standard TLM ainsi que de la librairie associée.

2.2.1 Les concepts de base de la modélisation TLM

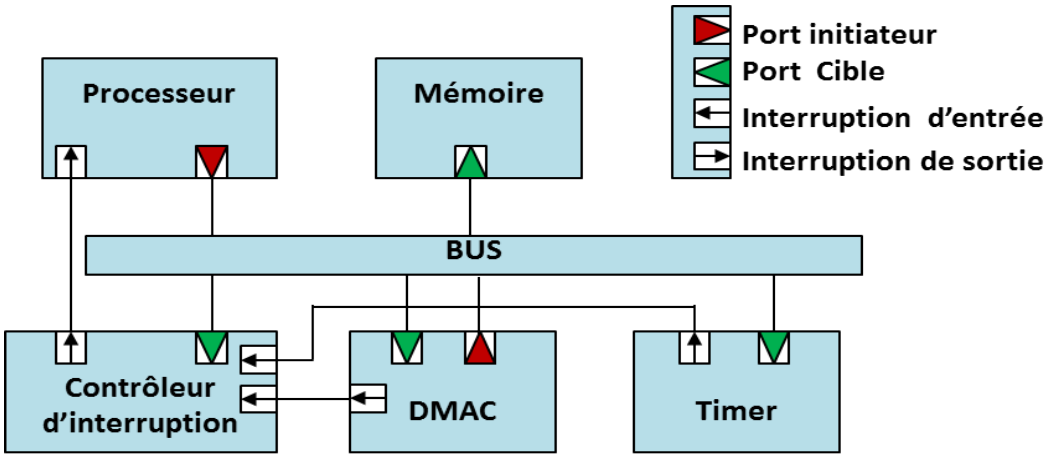


Figure II.2 – Exemple d'une plateforme TLM

Un exemple d'une plateforme TLM composée de 5 composants reliés par des connexions entre leurs ports est donné sur la Figure II.2. Le bus joue le rôle de canal de communication entre les composants.

Deux types de communication peuvent être distingués : une communication via des interruptions et une communication via des transactions.

La communication via des interruptions permet d'échanger des données de manière unidirectionnelle entre les composants par une connexion Point à Point. Chaque interruption est un

événement externe qui modifie le déroulement d'un programme de manière asynchrone. Comme illustré dans la Figure II.2, il existe deux types de port pour établir les interruptions : un port d'entrée et un autre de sortie.

La communication via des transactions consiste à un échange de données atomique entre un composant initiateur (ou maître) et un composant cible (ou esclave). L'initiateur peut initier une transaction alors que le composant cible est toujours en attente de recevoir une transaction. L'initiateur émet des transactions via un port initiateur. Ces transactions sont reçues par l'esclave par un port cible. Des composants peuvent avoir seulement des ports initiateurs comme les micro-processeurs (par exemple, le processeur de la Figure II.2), ou que des ports cibles (par exemple, la mémoire). Certains composants contiennent à la fois un port initiateur et un port cible comme le contrôleur d'accès direct à la mémoire (DMAC) de la Figure II.2. Ce contrôleur reçoit du processeur des paramètres de transfert par son port cible et exécute leur transfert via son port initiateur.

Les informations échangées par l'intermédiaire d'une transaction dépendent du protocole de bus utilisé. Cependant, certaines d'entre elles sont communes à tous les protocoles :

- Le type de transaction détermine le sens de l'échange de données, il correspond généralement à une lecture ou une écriture.
- L'adresse est représentée sous forme d'un nombre entier positif qui désigne le composant cible et un registre ou une adresse mémoire d'un composant interne.
- Un ensemble de données communiquées est envoyé ou reçu (notion de *payload*).
- Certaines méta-données supplémentaires peuvent compléter une transaction, y compris par exemple un statut de retour (erreur, succès, etc.), la durée de la transaction, des attributs de bus (comme un attribut de priorité).

La fonctionnalité principale de tous les bus est d'acheminer les transactions à leurs destinations identifiées par leurs adresses. Cette adresse est définie par le *mapping* global de l'espace mémoire supporté par le bus. En effet, ce *mapping* associe pour chaque port cible une zone de mémoire qui lui permet d'une part d'être identifié et d'autre part d'accéder à l'espace de mémoire locale ou aux différents registres du bloc matériel connectés à ce port.

Afin d'exécuter correctement le logiciel embarqué, ce *mapping* global des adresses et les adresses locales des registres (*offsets*) doivent être les mêmes que dans la puce finale (*Register Accuracy*). De plus, les données produites et échangées par les composants doivent également être identiques en type et en représentation binaire (*Data Accuracy*). Enfin, il est nécessaire que les interruptions correspondent logiquement aux interruptions finales. Par contre, compte tenu du modèle transactionnel, la précision temporelle des occurrences des interruptions par rapport à la puce finale n'est pas assurée. L'ensemble de ces exigences représente un contrat

entre le logiciel embarqué et le matériel. Ce contrat renforce la confiance d'un fonctionnement correct du logiciel embarqué sur la plateforme virtuelle.

2.2.2 TLM avec SystemC

Nous avons présenté les concepts communs du niveau transactionnel qui nécessitent un langage informatique concret pour leur mise en œuvre. Les langages de description de matériel traditionnels (VHDL et Verilog) n'ont pas été développés pour décrire des modèles TLM. En effet, ils ont été initialement ciblés à des niveaux d'abstraction inférieures. Par ailleurs, les langages de programmation comme C ou C ++ n'intègrent pas la possibilité de décrire simplement des architectures matérielles temporisées. Pour ces raisons, de nombreux langages sont apparus pour modéliser des architectures matérielles à un plus haut niveau d'abstraction que VHDL ou Verilog. D'une part, certains ont suivi une approche orientée matérielle pour étendre les langages de description de matériel existants avec des constructions et des concepts logiciels supplémentaires comme System Verilog [Ver03]. D'autre part, nous trouvons également l'approche orientée logicielle qui consiste à intégrer des fonctionnalités matérielles aux langages de programmation classiques. Nous citons les exemples de HANDELC [yTS02] et de SPECC [RAG02] qui appliquent cette approche au langage C. Ces initiatives ont eu un succès limité car elles nécessitent l'utilisation d'un compilateur spécifique et elles impliquent une dépendance vis-à-vis d'un seul vendeur d'outils de CAO.

SystemC [sys] est une librairie de classes C ++ développée par *Open SystemC Initiative* (OSCI) qui est une association indépendante dédiée à la définition d'un standard industriel ouvert et avancé pour la modélisation au niveau système, la conception et la vérification. Contrairement à d'autres langages qui nécessitent des outils spécifiques, SystemC utilise un compilateur C++ standard qui produit un exécutable contenant à la fois le modèle et le simulateur associé. Ainsi, SystemC peut être distribué comme toute autre librairie logicielle.

Le standard SystemC offre un ensemble de primitives pour la description des activités parallèles représentant le comportement physique des blocs matériels. Il propose également un environnement de simulation complet avec un ordonnanceur non-déterministe et non préemptif. Depuis 2011, ce standard intègre la modélisation transactionnelle TLM-2.0 (II.3) ce qui permet la validation fonctionnelle basée sur la simulation d'un modèle abstrait d'un SoC. Les composants d'une plateforme TLM sont ainsi modélisés comme des modules SystemC. Ces modules exposent des ports qui représentent certaines entités physiques qui peuvent être activées

en parallèle lors de l'exécution des comportements des modules pour obtenir la fonctionnalité attendue du système final.

2.3 Le standard OSCI TLM 2.0

Dans ce paragraphe, nous introduisons le standard TLM 2.0 et ses apports par rapport à la version 1.0. Comme il y a un besoin de simulation à un plus haut niveau d'abstraction, la norme OSCI TLM 1.0 devient moins applicable et conduit à des simulations qui ne sont pas assez rapides dès que la complexité du système à simuler augmente. L'absence d'un modèle standard d'interopérabilité présente une autre lacune de cette première norme. Afin de surmonter ces problèmes, une norme OSCI TLM 2.0 a été développée. Son objectif est de fournir un modèle de plateforme disponible dans les premières phases de conception pour le développement de logiciels, pour aider à l'intégration logicielle/matérielle et pour évaluer des performances logicielles et matérielles. La norme comporte un ensemble d'interfaces de base, de protocoles et de classes pour les ports de communication, les données, et d'autres services. Ces interfaces et ces classes sont particulièrement adaptées à la conception de modèles de systèmes construits autour de bus basés sur un *mapping* mémoire. Afin de maximiser l'interopérabilité, ce standard OSCI propose aussi des mécanismes d'extension pour modéliser des interconnexions basées sur un protocole spécifique de type *memory-mapped* ou non.

Les classes TLM 2.0 sont disposées en couches au-dessus de la bibliothèque de classe SystemC et incluent les classes développées dans la norme TLM 1.0 comme illustré dans la Figure II.3. En plus des classes utilitaires, la norme 2.0 implique une couche d'interopérabilité spécifique pour la modélisation de bus (voir Figure II.3). Cette couche est constituée par la classe *generic paylaod*, les phases de protocole de base, les classes de sockets initiateur et cible et les interfaces cœurs du standard TLM-2. Ainsi, la couche d'interopérabilité permet de standardiser les transactions TLM ainsi que la communication des informations de synchronisation entre les modèles.

TLM 2.0 définit deux principaux styles de codage pour créer des modèles avec un certain degré de précision relatif à la synchronisation et à la communication. Les deux styles, *Loosely Timed* (LT) et *Approximately Timed* (AT), correspondent respectivement aux deux sous-niveaux initialement définis dans le standard TLM *Programmer'sView* (PV) et PV+Time (PVT).

Le style LT emploie principalement l'interface bloquante de communication. Cette interface ne considère que deux instants dans le temps : le début et la fin d'une transaction, associés respectivement à l'appel et au retour de la fonction de transport bloquante. En effet, le premier point de synchronisation marque le début de la demande et le second marque le début de la

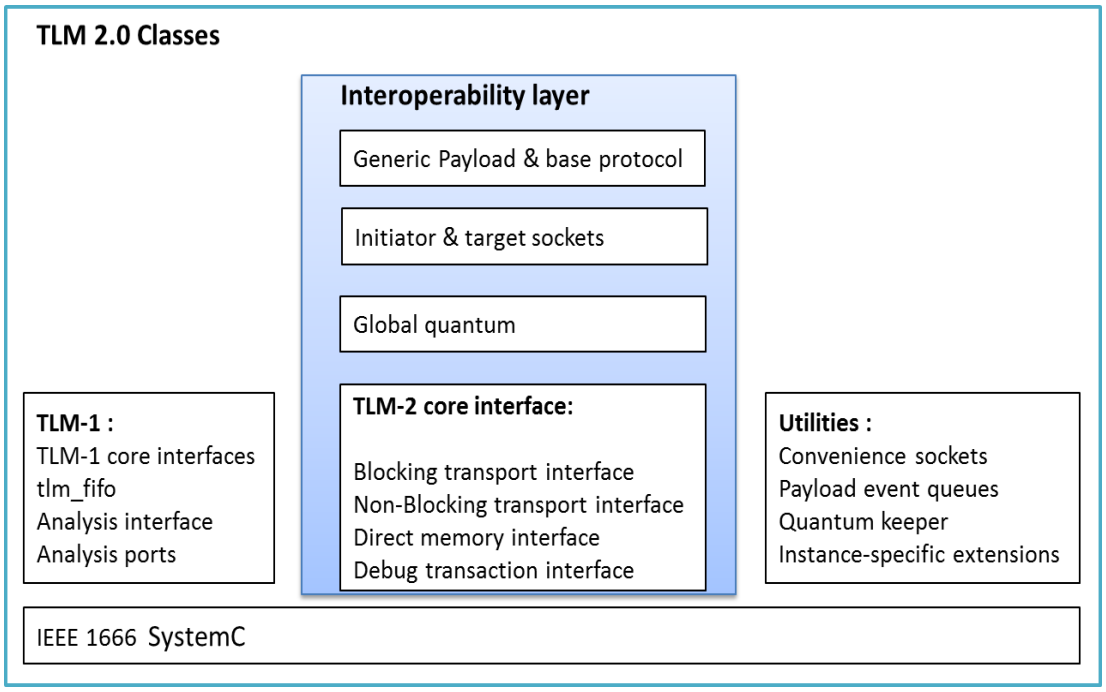


Figure II.3 – Aperçu du standard OSCI TLM 2.0 [sys]

réaction. Ces deux points de synchronisation peuvent se produire dans un même temps de simulation ou à des moments différents.

Par ailleurs, le style AT intègre plus de précision de synchronisation pour les transactions échangées entre les composants d'un système en intégrant plusieurs points de synchronisation dans chaque transaction. L'interface adaptée à ce style est l'interface de communication non-bloquante qui permet de décomposer chaque transaction en plusieurs phases. Il est approprié pour l'exploration architecturale et l'analyse de la performance.

2.4 Présentation générale du standard IEEE 1685 (IP-XACT)

IP-XACT [IP-] est un standard initialement défini par le consortium SPIRIT puis repris dans le cadre de l'initiative Accellera [1]. Ce standard propose un langage basé sur le format XML (*eXtensible Markup Language*) pour la description structurelle des architectures matérielles. En effet, il permet la description et l'assemblage des blocs IP (*Intellectual Property*) d'une plateforme matérielle modélisant un système sur puce complet. Ces IP peuvent être réutilisées et intégrées dans différents environnements de développement. Les premières versions du standard permettent la description électronique des architectures au niveau RTL. Depuis 2008, IP-XACT

1. www.accellera.org

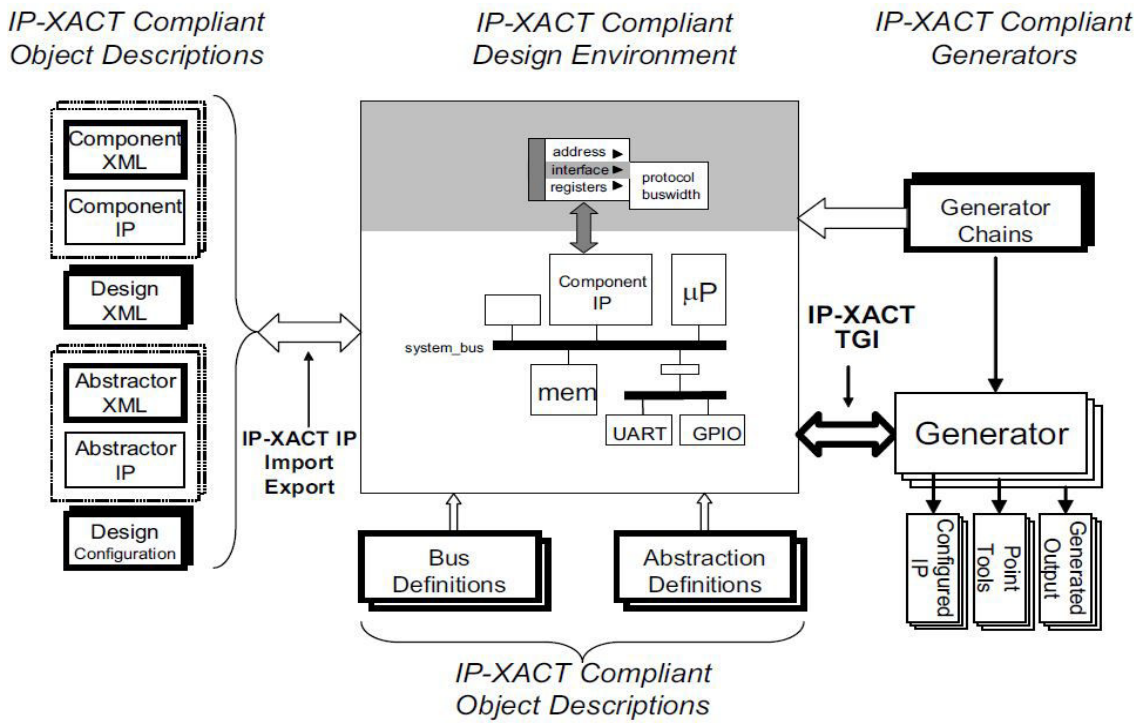


Figure II.4 – Étapes d'un flot de conception IP-XACT [IP-]

(version 1.4) offre un support à la modélisation au niveau TLM, via la définition de port transactionnel.

Comme l'illustre la Figure II.4, le flot de conception dans IP-XACT repose sur trois étapes principales. La première étape consiste à rassembler tous les éléments d'une bibliothèque d'IPs dans des fichiers XML conformément aux schémas XSD. Ces éléments peuvent être les composants eux-mêmes et leurs propriétés (interface, espace d'adressage, vendeur, version, etc.) ainsi que des fichiers qui pointent vers des codes sources VHDL, SystemC, etc. Dans la deuxième étape, un assemblage des instances de ces IPs est réalisé afin de construire un modèle d'une architecture (*Design*). La dernière étape permet l'intégration et la connexion avec des outils externes. Elle est définie par des éléments IP-XACT de types *generator* et *generator Chain*. Les *generator* extraient les données des modèles d'architecture à travers une API fournie par le standard et nommée TGI (*Tight Generator Interface*). Un *generator Chain* permet la description des enchaînements des outils à travers le séquencement des éléments de type *generator*. Nous présentons maintenant brièvement les composants de base définis dans le standard IP-XACT et qui permettent de décrire un système.

Tout périphérique, processeur ou composant de stockage/d'interconnexion est décrit par

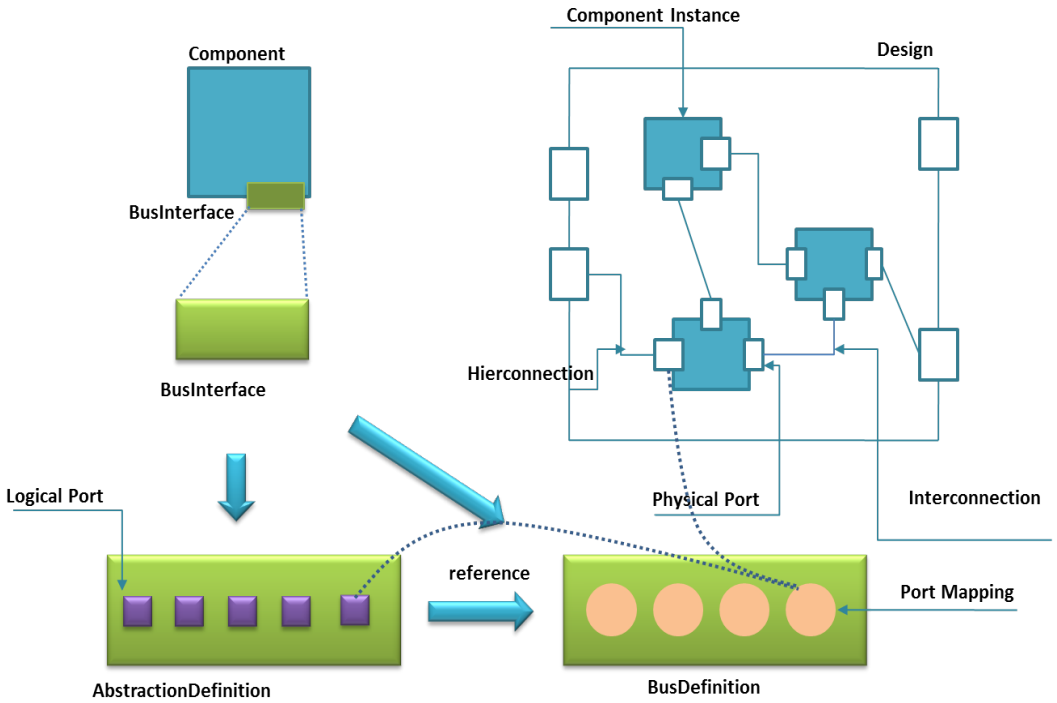


Figure II.5 – Composants de base du standard IP-XACT [MRA10]

l'élément *component* qui est l'élément central du standard. Deux types de *component* peuvent être distingués : (1) statique lorsque le *component* est instancié sans aucune modification des valeurs de ses paramètres définis à la création et (2) configurable dans le cas où un paramétrage est nécessaire lors de l'instanciation du *component*. Un *bus Interface* permet la modélisation de l'interface de communication du *component*. Indépendant du niveau d'abstraction considéré, il décrit les attributs structurels associés au bus. Il référence un élément de type *AbstractionDefinition*. Cet élément *AbstractionDefinition* englobe les informations détaillées pour la définition de bus. Plusieurs *AbstractionDefinition* sont possibles pour définir un même bus, par exemple une qui correspond au niveau RTL et une autre au niveau TLM. Une *AbstractionDefinition* a deux attributs obligatoires, son *busType* et ses ports. L'attribut *busType* fait référence à l'élément *BusDefinition*. Cet élément permet de préciser des aspects de haut niveau d'une interface de communication comme les protocoles de communication (par adressage ou directe).

Au niveau de l'élément *Design*, les composants sont instanciés et leurs interconnexions sont établies à travers les éléments décrits ci-dessus et illustrés sur la Figure II.5.

Des propriétés fonctionnelles et extra-fonctionnelles peuvent être intégrées dans une définition de *component* ou d'un *Design* en utilisant des *Vendor Extensions* [Ven]. Le format et le type d'information contenus dans une *Vendor Extension* propriétaire ne sont pas définis

dans la norme IP-XACT. Ainsi, une implémentation des mécanismes d'analyse et d'interprétation spécifiques sont nécessaires afin de partager ces informations entre différents outils et fournisseurs.

2.5 La réduction de la puissance dans les SoC

Avant d'aborder les techniques de réduction de consommation de la puissance dissipée, nous commençons par rappeler les composantes de cette puissance.

La puissance totale consommée au sein d'un circuit peut se diviser en deux composantes : (1) une consommation statique due principalement à des courants de fuite, et (2) une consommation dynamique résultant de l'activité de commutation (*switching activity*) des circuits. Lors de l'optimisation de la consommation, ces deux composantes doivent être prises en compte et toutes les deux requièrent des techniques de gestion et de réduction différentes.

2.5.1 Puissance statique

La puissance statique est une consommation électrique devenue non négligeable avec les avancées de la technologie, elle est due aux courants de fuite. Cette puissance est le produit de la tension d'alimentation par le courant de fuite comme l'indique l'équation II.1. Les courants de fuite résultent du courant « sous-seuil » et du « courant-inverse ». Ils dépendent fortement de la technologie utilisée et de la longueur du canal du transistor.

$$P_{Statique}(t) = V * I_{fuite} \tag{II.1}$$

2.5.2 Puissance Dynamique

La puissance dynamique est liée aux deux facteurs prépondérants : le courant de court-circuit et le courant de commutation. Ces deux courants apparaissent lors de la commutation des états des transistors. La puissance dynamique, comme indiqué sur l'équation suivante, est le produit du taux d'activité du circuit(α), de la capacité équivalente du circuit (C), du carré de la tension d'alimentation (V) et de la fréquence d'horloge(f_{clock}). Ces paramètres dépendent de la technologie utilisée ainsi que de l'application exécutée.

$$P_{Dynamique}(t) = \alpha * C * V^2(t) * f_{clock}(t) \tag{II.2}$$

2.5.3 Approches classiques de réduction de puissance dans un SoC

Au niveau de l'architecture matérielle, les techniques classiquement employées dans les SoC pour gérer la puissance dissipée ciblent soit la consommation de puissance statique, soit celle relative à la puissance dynamique, soit la puissance totale. Ce paragraphe présente les techniques les plus utilisées : le *power gating*, le *clock gating* et l'ajustement multi-tensions.

2.5.3.1 *Power gating* :

La technique de *power gating* appelée également DPM (*Dynamic Power Management*) ou encore DPS (*Dynamic Power Switching*) vise à couper l'alimentation des blocs d'un système non utilisés à un instant donné. Elle est la seule permettant d'annuler complètement la puissance statique. Pour appliquer cette technique, les concepteurs partitionnent généralement le SoC en *Power Domains* (Figure II.7). Un *Power Domain* (PD) est un ensemble de blocs alimentés par la même source d'alimentation et tel que le contrôle de type *on/off* sur cette source d'alimentation est commune à tous les blocs du domaine. La stratégie de base du *power gating* est de fournir deux modes de puissance : mode actif (*ON*) et mode inactif (*OFF*). L'objectif est de basculer d'un mode à autre en temps opportun et de manière appropriée afin de maximiser les économies d'énergie tout en minimisant l'impact sur les performances. En effet, utiliser en pratique cette technique engendre deux coûts :

- un coût temporel : des latences importantes de transition d'un mode *ON* à *OFF* et inversement peuvent apparaître.
- un coût énergétique : l'énergie nécessaire en particulier pour remonter les niveaux de tension lors du passage de l'état *OFF* à *ON*.

Par conséquent, pour que l'application de cette technique permette des gains en énergie, il faut que le temps pendant lequel le bloc peut être mis en mode inactif soit suffisamment grand pour que les pénalités en énergie dues aux changements d'états ne soient pas supérieures à l'énergie dissipée si le bloc restait sous tension. Ce temps minimum est classiquement appelé le *Break Event Time* [BBDM00]. L'évaluation de ce temps T_{be} est exprimée dans [WWL03] en fonction des paramètres E_{sd} : énergie de passage à l'état OFF, E_{wu} : énergie de réveil (*wake up*) pour passer à l'état ON, P_s : puissance statique, P_w : somme des puissances statique et dynamique, T_{sd} : délai pour passer à l'état OFF et T_{wu} : délai pour passer à l'état ON. L'expression de T_{be} est rappelée dans l'équation II.3.

$$T_{be} = \frac{E_{sd} + E_{wu} - P_S * (T_{sd} + T_{wu})}{(P_W - P_S)} \tag{II.3}$$

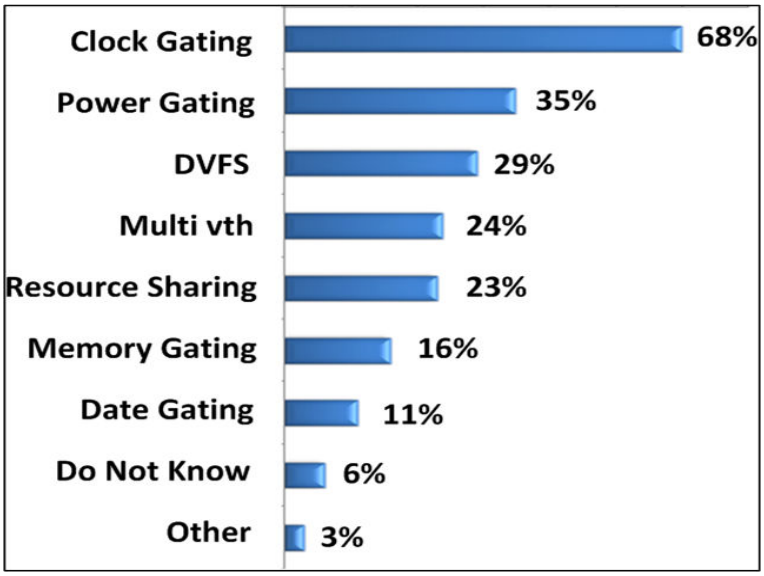


Figure II.6 – Les 10 meilleures techniques pour réduire la consommation d'énergie [Clk]

2.5.3.2 *Clock gating* :

La consommation de la puissance dynamique est due principalement à la distribution d'horloge. En effet, l'arbre d'horloge consomme entre 30% et 70% de la consommation de puissance dynamique [FAGS07]. La technique la plus courante pour réduire cette consommation, consiste à inhiber l'horloge sur les blocs qui ne sont pas utilisés à un instant donné. Cette technique appelée *clock gating* est fortement utilisée comme illustré dans la Figure II.6. L'application de cette technique nécessite une décomposition du SoC en *Clock Domains* comme le montre la Figure II.7. Un *Clock Domain* est un ensemble de blocs alimentés par la même source d'horloge. Ainsi, le *clock gating* permet de stopper toute activité dans le *Clock Domain* concerné et par conséquent, d'annuler la consommation dynamique associée.

Contrairement aux pénalités produites en appliquant le *power gating*, celles relatives à l'extinction de l'horloge et sa remise en service sont très faibles. La difficulté réside plutôt dans la conception de l'arbre d'horloge qui doit assurer une synchronisation correcte des horloges sur les différents blocs du système.

2.5.3.3 Ajustement Multi-Tensions :

Les puissances dynamique et statique sont proportionnelles respectivement à V^2 et V. Ainsi, réduire la tension d'alimentation de certains blocs d'un système sur puce affecte directement

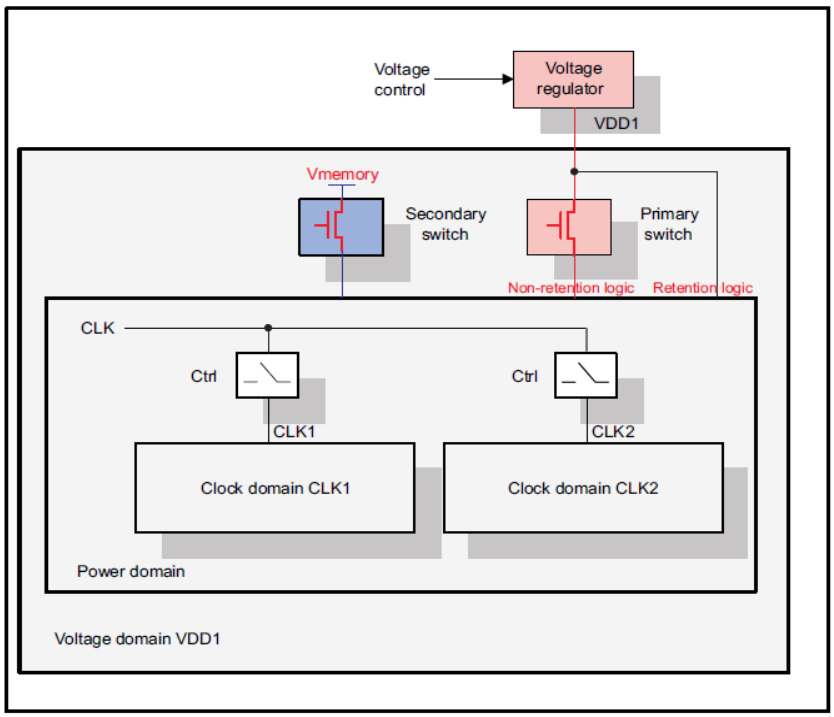


Figure II.7 – Exemple de décomposition en *Power*, *Clock* et *Voltage Domains* [OMAa]

sa puissance dissipée totale. Cependant, tous les blocs ne nécessitent pas réellement la même tension de fonctionnement. Par exemple, un processeur demande en général une tension d'alimentation pouvant être relativement élevée suivant la complexité des tâches à exécuter mais, une tension relativement plus basse peut être suffisante pour un périphérique comme un contrôleur USB pour assurer son fonctionnement. Par conséquent, le processeur et le contrôleur USB peuvent être inclus dans deux domaines de tension différents. Un domaine de tension ou *Voltage Domain* est un ensemble de blocs ou sous-systèmes alimentés et contrôlés par le même régulateur de tension (*Voltage Regulator*) comme illustré sur la Figure II.7. Dans la suite, nous présentons une classification de stratégies d'ajustement en multi-tensions définie dans [FAGS07]. Quatre types de stratégies sont distingués :

Ajustement statique de tension (*Static Voltage Scaling* - SVS) : Selon cette stratégie, les différents blocs ou sous-systèmes sont alimentés par différentes tensions d'alimentation fixes. Ces tensions sont ajustées pour permettre un fonctionnement correct de chaque sous-système suivant la fréquence d'horloge fixée pour ce sous-système. Rappelons que le temps de réponse d'une porte logique dépend de sa tension d'alimentation [CSB92], aussi chaque tension doit être déterminée pour garantir la fréquence associée à chaque sous-système.

Ajustement multi-niveaux de tension (*Multi-level Voltage Scaling* - MVS) : Il s'agit d'une extension de l'ajustement statique de tension dans laquelle un bloc ou un sous-système bascule entre deux ou plusieurs niveaux de tension fixes et discrets.

Ajustement dynamique de la fréquence et de tension (*Dynamic Volatge Frequency Scaling* - DVFS) : Dans ce type d'ajustement, un contrôleur DVFS adapte la puissance dynamique en fonction du niveau de performance souhaité par le ou les bloc(s) qu'il contrôle. Cette technique permet de (1) réduire la fréquence des blocs contrôlés lorsqu'ils sont sous-utilisés, (2) augmenter la fréquence des blocs contrôlés au-dessus de leur fréquence maximale (*over clocking*) dans le cas où il y a un besoin ponctuel d'une puissance accrue, et (3) ajuster la valeur de tension relativement à la fréquence utilisée. Cette adaptation dynamique se fait en réglant les valeurs de couples de (voltage/fréquence) appelés *Operating Performance Points* (OPPs). Les valeurs dépendent à la fois de l'architecture matérielle du bloc et de son comportement fonctionnel (complexité de l'application). Dans ces conditions, le contrôleur DVFS agit sur la consommation de puissance dynamique et la puissance statique du SoC tout en répondant aux exigences de l'application. En pratique, il peut y avoir plusieurs contrôleurs DVFS dans un SoC ce qui implique un partitionnement des blocs fonctionnels de l'architecture suivant ces contrôleurs (*Clock Domains*/ *Voltage Domains*).

Ajustement adaptatif de tension (*Adaptive Voltage Scaling* - AVS) : L'AVS est une technique utilisée directement au niveau matériel pour effectuer le contrôle de la tension d'alimentation. Son objectif est d'adapter au mieux la tension d'alimentation par rapport à la fréquence souhaitée en tenant compte en même temps de la température du circuit à un instant donné et de la variabilité du processus de fabrication du circuit. La variabilité du processus de fabrication induit des caractéristiques différentes entre circuits et pour deux circuits différents la valeur de la tension minimum permettant de faire fonctionner le circuit à une fréquence donnée peut varier. Dans ce contexte, l'AVS adapte la tension en fonction des caractéristiques du circuit. De plus, à température élevée, la tension peut être réduite si la fréquence de fonctionnement est faible alors que cette tension doit être augmentée pour faire fonctionner le circuit à fréquence élevée [Vin13]. À de basses températures le phénomène s'inverse. L'ajustement adaptatif modifie la tension d'alimentation suivant la température pour s'adapter au plus juste à la situation et ainsi, réduire la consommation d'énergie statique et dynamique.

Le problème de la gestion de puissance utilisant une ou plusieurs des techniques présentées ci-dessus a fait l'objet de très nombreuses études. Dans le domaine de l'ordonnancement temps

réel de tâches sur une architecture multi-cœurs de très nombreuses études ont été publiées. Nous citons à titre d'exemple ici la thèse de Vincent Nelis [Nel10] qui a proposé un algorithme optimal d'ordonnancement capable d'exploiter le DVFS par l'utilisation des *slack times* produits par les tâches qui terminent leur exécution avant leur WCET (*Worst Case Execution Time*). Le problème de l'exploitation du DPM pour minimiser l'énergie statique dans un contexte d'ordonnancement temps réel multiprocesseur est abordé par exemple dans [LJP13]. Une approche mixte utilisant DPM et DVFS est considérée dans [BBA11]. Toutes ces techniques tentent d'adapter des techniques classiques d'ordonnancement mono ou multiprocesseurs pour intégrer le critère de minimisation de la puissance consommée.

Dans le cas des applications orientées flots de données, des techniques de *power management* hors ligne sont généralement considérées dans la littérature. Classiquement, le problème du *power management* pour des graphes de tâches de type *Khan Process Network* est exprimé sous la forme d'un problème d'optimisation convexe sur lequel des techniques classiques de résolution de tels problèmes sont appliquées (par exemple, [NMM+11] et [LW13]). Ici, le graphe de tâches décrit un scénario applicatif spécifique sur lequel est formulé le problème d'optimisation permettant d'extraire une solution spécifique à ce scénario. Avec des systèmes capables de télécharger un grand nombre d'applications ayant des profils très différents, cette approche d'optimisation hors ligne apparaît ne pas être adaptée. Ainsi, ces techniques ne sont généralement pas considérées dans la pratique pour de tels systèmes. Une autre difficulté relative aux approches citées ci-dessus est liée au fait qu'un système construit autour d'une architecture de SoC contient de très nombreux composants autres que des CPU qui sont également à l'origine d'une forte consommation d'énergie (par exemple, un GPU). D'autres approches que celles brièvement présentées ci-dessus sont également considérées dans la pratique. Nous pouvons distinguer :

- La gestion du DVFS basée sur l'analyse en ligne de la demande en performance de l'application.
- La gestion directe des états ON et OFF des composants en observant leurs états fonctionnels.

2.5.4 Gestion du DVFS en ligne

Dans de très nombreux systèmes embarqués, en particulier basés sur un système d'exploitation tel que Linux, la gestion du DVFS concerne le ou les cœurs de processeur et s'effectue via des OPP (*Operating Performance Point*) fixés a priori. Chaque OPP définit un couple <voltage,fréquence> qui peut être sélectionné au moment de l'exécution. Les algorithmes d'optimi-

sation qui calculeraient un ensemble de valeurs optimales des OPP pour une application donnée sont en général trop complexes pour être implémentés en ligne. Par exemple, dans [DKV+15], le modèle est basé sur un algorithme de régression qui cherche à maximiser un maximum de vraisemblance. C'est effectivement une approche assez complexe qui a peu de chance d'être utilisée en ligne. Il semble plus adapté d'utiliser une heuristique et de l'activer à chaque fois que l'application change ou change d'état fonctionnel. Cependant, la plupart des solutions qui opèrent en ligne préfèrent s'adapter à la charge de travail, par exemple grâce à un asservissement ou à une prédiction de la charge de travail par processeur. Cette approche est plus simple et plus générique puisqu'elle ne dépend pas de l'application exécutée. Les OPP peuvent être sélectionnés aussi dynamiquement en fonction du niveau de performance souhaité.

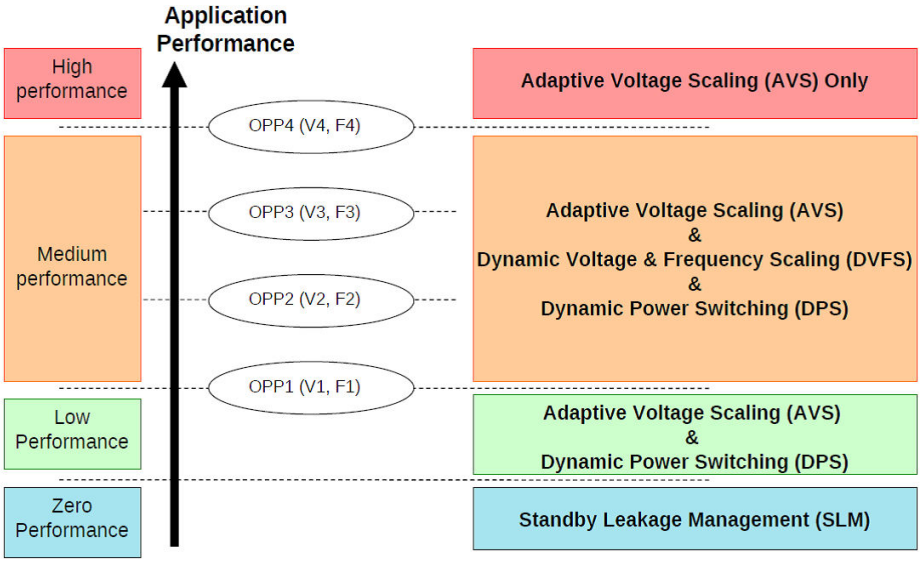


Figure II.8 – Impact des OPP dans l'activation des techniques de *power management* d'un OMAP44xx [AM]

Par exemple, dans un OMAP44xx, les différents OPP permettent de mettre en œuvre ou non les différentes techniques matérielles de gestion de puissance dissipée présentées sur la Figure II.8. Dans cette Figure, le couple <voltage,fréquence> noté OPP (V1,F1) correspond à la fréquence la plus basse déclarée pour le système. A cette fréquence la capacité de calcul du circuit est en général loin d'être exploitée totalement. Par exemple, cet OPP peut être sélectionné lorsque le système est en mode veille. De plus, il est possible avec cet OPP d'appliquer également et localement la technique de DPS. Cette fréquence peut correspondre à celle fournie par un oscillateur externe basse fréquence aussi, il n'est pas possible d'avoir plusieurs OPP

avec cette fréquence externe. Avec les OPP (V2,F2) et OPP (V3,F3), les *Digital Phase Locked Loop* (DPLL) internes au circuit peuvent être configurées suivant ces fréquences de sortie ce qui revient à activer le DVFS en plus du DPS et de l'AVS. Enfin, à la fréquence la plus haute, il n'est plus possible d'effectuer du DVFS, le DPS n'a a priori pas de sens car la performance maximum est visée et seul l'AVS peut encore être appliqué.

2.5.5 Gestion du DPM en ligne

Dans les systèmes basés sur Linux, la stratégie de gestion de l'alimentation est principalement organisée en utilisant une vue centrée sur le CPU. Le niveau de performance, défini par les OPPs, est ajusté dynamiquement en fonction de la charge de traitement des cœurs de processeurs où la charge est estimée par le système d'exploitation [KVG+12]. Pour cela, dans Linux, la fonction CPU_idle() est implémentée de sorte à renseigner le système sur l'état fonctionnel du CPU concerné. Cette fonction devient active dès que le CPU n'a plus de tâche à exécuter. A partir de cet état *Idle* du CPU, il est possible d'appliquer une stratégie de DPM pour éteindre le CPU correspondant. Comme déjà indiqué, le système devrait normalement tenir compte du délai T_{be} pour activer et désactiver les blocs IP concernés sous peine d'un bilan énergétique négatif. Pour ces systèmes centrés CPU, cette approche de gestion de l'énergie peut être efficace. Par contre, pour des systèmes intégrant un très grand nombre de blocs fonctionnels, l'approche de gestion via Linux peut s'avérer peu efficace parce qu'il n'est pas possible au niveau système ou au niveau applicatif d'être suffisamment réactif dans la gestion de puissance dissipée de chaque bloc en fonction de son état fonctionnel. C'est pourquoi les mécanismes au niveau matériel sont souvent mis en œuvre pour gérer au plus près l'état d'alimentation de ces blocs. Par ailleurs, au niveau logiciel, des mécanismes ont été par exemple proposés dans *Android* pour permettre le maintien en état *ON* de certains sous-systèmes (*Wake Locks*) et éviter ainsi des prises de décision jugées inopportunes dans les couches matérielles de gestion de puissance [DBN12].

Pour conclure, les techniques d'analyse hors ligne en vue de l'optimisation de l'énergie dans des SoC complexes et sur des scénarios applicatifs ne sont plus véritablement considérées dans la pratique parce que la diversité des scénarios est trop importante et le chargement dynamique d'applications sur un système rend inappropriée l'utilisation de ces approches d'optimisation. Il ressort ainsi une exigence dans nos travaux qui est que la ou les techniques d'optimisation de la consommation de puissance doivent s'adapter à tout type d'application dès lors qu'elle implique un niveau significatif de consommation d'énergie. Ce dernier point signifie qu'une application qui s'exécute (périodiquement) en un temps très réduit ne nécessite pas spécialement d'optimisation

dynamique en puissance/énergie, ce qui du reste risquerait fort d'être inefficace du fait de prises de décisions du *power manager* qui pourraient être inappropriées par rapport au rythme des changements d'états fonctionnels de ce type d'application. Par conséquent, nous considérons dans nos travaux des techniques de *power management* en ligne qui visent à réduire au mieux la consommation d'énergie et à maximiser la qualité de service (voir chapitre V).

2.6 Standards utilisés dans les approches de réduction de puissance

Les langages de description matérielle (*Hardware Description Language* - HDL) ont été développés pour modéliser la structure et le fonctionnement des circuits intégrés. Cependant, ces langages n'ont pas une représentation implicite pour mettre en œuvre tous les éléments qui sont impliqués dans les techniques de réduction de puissance. Répondant à ce besoin, un groupe de quelques industriels au sein d'*Accellera Systems Initiative* a élaboré une norme UPF (*Unified Power Format*). La première version d'UPF a été publiée en 2007 et, dans la même année, elle a été transférée à l'IEEE afin de créer une nouvelle norme IEEE. En 2009, l'IEEE publie la première norme pour la spécification d'une structure de contrôle de puissance appliquée à un SoC, la norme «IEEE-1801» [upf]. Cette norme est appelée aussi UPF 2.0. En parallèle, un second format est proposé CPF (*Common Power Format*) qui est plutôt géré par *Silicon Integration Initiative* (Si2) [cpf11]. Les formats CPF et UPF sont basées sur un langage de script TCL (*Tool Command Language*).

Ces deux formats définissent un langage et une sémantique de simulation permettant de spécifier la façon dont les sources d'alimentation sont fournies, distribuées et dynamiquement gérées dans un système embarqué à faible consommation : un *power intent*. UPF et CPF décrivent ainsi les concepts pour une conception faible consommation ainsi que les informations nécessaires au contrôle de cette consommation, indépendamment de la spécification fonctionnelle. En effet, ils sont basés sur une approche de séparation des aspects fonctionnels et non-fonctionnels afin de supporter une démarche où les aspects fonctionnels et performance d'un modèle sont étudiés et validés dans un premier temps avant d'être optimisé en consommation et ce en gardant comme référence le modèle fonctionnel. Ces spécifications UPF ou CPF sont décrites dans un même format utilisé à partir du niveau RTL jusqu'au niveau *Layout* comme le montre la Figure II.9. En conséquence, ils sont utilisés à la fois durant la simulation, la synthèse et le placement/routage, évitant ainsi les possibles erreurs lors de changement de niveau. Dans la suite, nous nous focalisons sur le standard UPF en définissant ses principaux concepts, illustrés sur la Figure II.10, pour décrire un *Power Intent* d'un SoC.

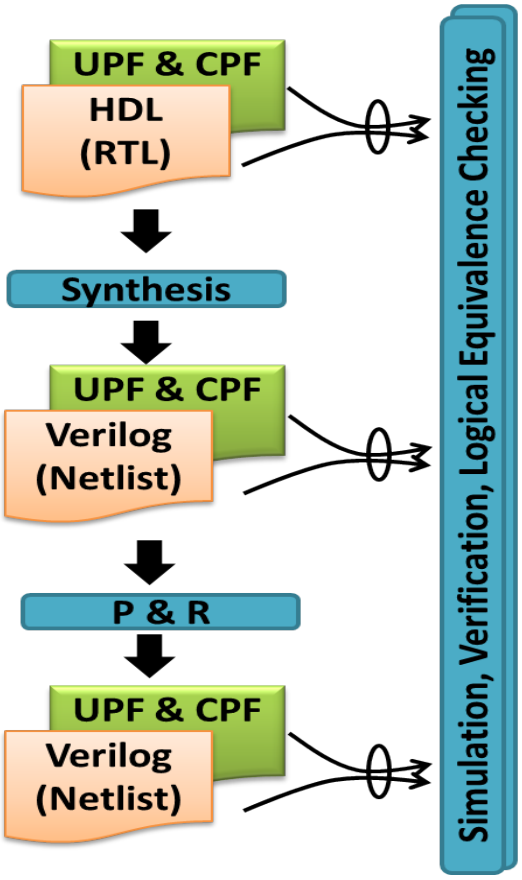


Figure II.9 – Flot UPF à partir du niveau RTL [upf]

Comme le montre la Figure II.10, l'architecture matérielle est partitionnée en différents *Power Domains* (PD) qui regroupent un sous-ensemble de composants du système recevant leur alimentation depuis les mêmes rails d'alimentation (*Supply Nets*). Ainsi, chaque PD peut être contrôlé individuellement. Il peut être mis dans différents états de fonctionnement appelés *Power States*. Les *Power States* sont définis par l'état des *Power Switches* insérés sur les *Supply Nets*. Par exemple, le bloc *Receiver* de la Figure II.10 est inclus dans le PD (PD_Receiver) alimenté par le rail d'alimentation VDD_LOW. L'état de ce PD dépend de l'état du *power switch* $S_{receiver}$.

Un des problèmes majeurs lors de l'application des techniques de réduction de puissance provient des interfaces de communication qu'il est nécessaire de définir entre les PD. Les *level shifters* (LS), les cellules d'isolation, et les registres de rétention représentent ces interfaces additionnelles. Les LS sont introduits entre deux PD alimentés avec des tensions différentes afin d'ajuster la tension des signaux logiques échangés. Ceci permet d'éviter les erreurs logiques

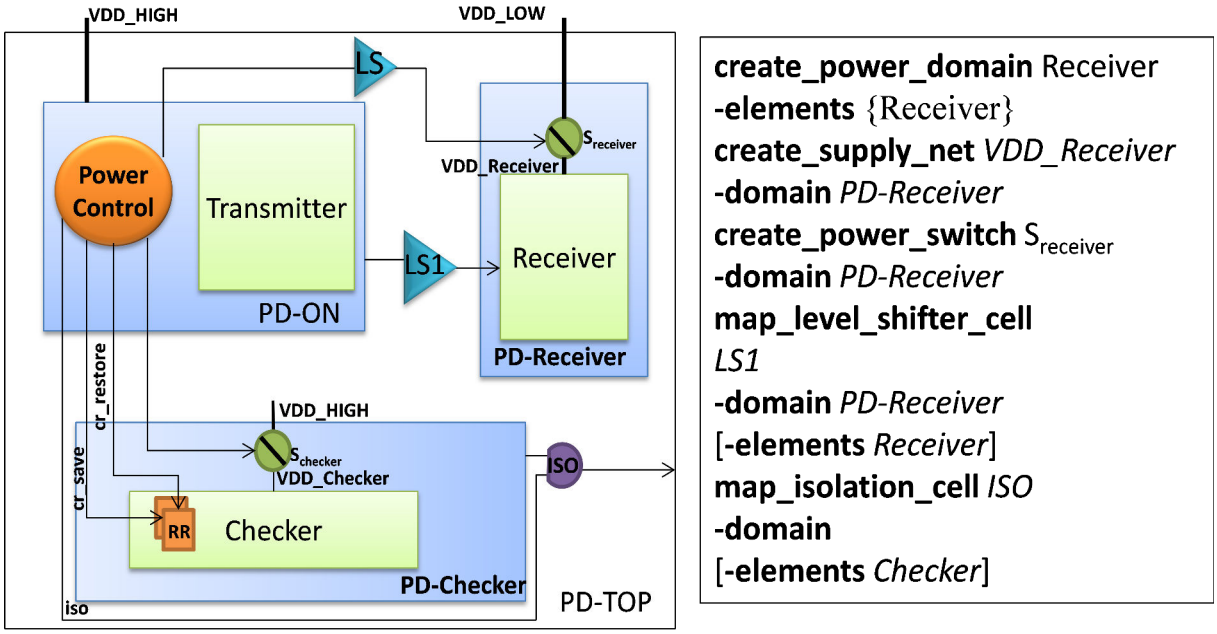


Figure II.10 – Exemples des concepts définis dans UPF

fonctionnelles qui peuvent se produire lorsqu'un bloc alimenté par une tension faible transmet un signal à destination d'un bloc alimenté par une tension plus élevée. Une autre interface spécifique est nécessaire lorsqu'une connexion est établie entre un PD à l'état inactif (*OFF*) et un autre PD à l'état actif (*ON*). En effet, l'état logique d'une sortie d'un bloc du PD à l'état OFF peut être indéterminé ce qui peut induire des fonctionnements incorrects. Pour remédier à ce problème, il est nécessaire d'introduire des cellules d'isolation (*Isolation Cells*) entre les deux PD de manière à garantir un état déterminé en sortie du *power domain* à l'état OFF. Par ailleurs, lorsqu'un PD passe de l'état ON à l'état OFF ou lors de la réception d'un signal de *Reset*, les états des registres ou mémoires internes des IPs sont perdus ce qui peut provoquer des erreurs fonctionnelles. Pour cela, la norme UPF définit des *Retention Registers* qui sont utilisés afin de sauvegarder l'état des registres ou mémoires lors d'une transition de l'état *ON* à l'état *OFF* d'un PD et de les restaurer lors d'une transition de l'état OFF à l'état ON. Un exemple de ces registres est présenté dans la Figure II.10 : RR sont des registres de rétention et les signaux cr_save et cr_restaure permettent leur contrôle. Les commandes TCL suivant le format UPF permettant la description de ces mécanismes sont indiquées à droite dans la Figure II.10.

Tous ces éléments nécessaires à la spécification de l'architecture *power* (i.e. le *power intent*) doivent être contrôlés. C'est le rôle du *Power Controller* qui permet la mise en place de la stra-

	VDD_HIGH	VDD_LOW	VDD_Checker	VDD_Receiver
Full Run	High_Voltage	Low_Volatge	ON	ON
Boot	High_Voltage	Low_Volatge	ON	OFF
Sleep	OFF	OFF	OFF	OFF

Figure II.11 – Exemple d'une *Power State Table*

tégie de gestion de puissance en spécifiant une interface de contrôle entre le modèle fonctionnel et les composants introduits pour agir sur la consommation de puissance du modèle fonctionnel. Ainsi, a priori un *Power Controller* doit être toujours dans un état actif afin de permettre à tout moment la mise hors/sous tension de tout bloc fonctionnel en liaison avec l'application à exécuter. Pour cela, il fait partie d'un PD de type *Always-ON* c'est-à-dire toujours alimenté. La stratégie de gestion de puissance est incarnée dans UPF par une table des états de puissance du système (*Power State Table* - PST) où les colonnes présentent les modes d'alimentation de chacun des PD de l'architecture tandis que chaque ligne correspond à un mode global d'alimentation du système. Ainsi, une ligne d'une PST décrit un ensemble de fonctionnalités activées correspondant à un scénario spécifique du système. La Figure II.11 est un exemple de PST pour l'architecture *power* illustrée sur la Figure II.10. Cette PST définit trois états globaux *Full Run*, *Boot* et *Sleep* en spécifiant une combinaison particulière d'états de *Power Domains*. Il est important enfin de noter que la PST définit une stratégie statique de gestion des modes *power*, cette stratégie ainsi que ses interfaces de contrôle avec le niveau applicatif ou logiciel système doivent être définies par le concepteur. La transition d'un mode global d'alimentation à un autre implique une séquence de contrôle qui doit être exécutée par un contrôleur. Par exemple, éteindre un PD implique d'activer en premier la rétention avant de couper l'horloge puis, d'activer les cellules d'isolation en sortie et enfin de mettre en mode *OFF* le *power switch* correspondant. Pour effectuer ces séquences de contrôle correctement, une table des transitions légales entre des modes globaux d'alimentation doit également être spécifiée en UPF.

Par conséquent , la norme UPF 2.0 propose des éléments permettant d'agir efficacement sur la réduction de la puissance, mais il y a encore quelques points où ce format a besoin d'être amélioré. Parmi ces points, nous citons par exemple :

1. La définition d'un modèle de consommation qui représente un bloc fonctionnel permettant d'extraire et de gérer facilement les informations non-fonctionnelles de ce bloc ;
2. L'introduction de *power state* composites qui peuvent être intéressants pour représenter

les systèmes composés de plusieurs sous-systèmes ;

3. La définition d'éléments permettant de décrire la structure et la distribution d'horloge dans un système autrement dit tous les éléments nécessaires pour modéliser et contrôler l'arbre d'horloge au niveau système (*Clock Domain*, *Clock Source*, etc.), leurs contraintes ainsi que les dépendances avec les PDs dont leurs caractéristiques sont définies en UPF 2.0.

Les deux premières lacunes sont en cours d'étude afin de produire la version UPF 3.0 [new] pour être plus efficace et réutilisable. Le troisième point fait l'objet de nos travaux et la solution proposée peut être considérée comme une contribution possible pour étendre ce format UPF 2.0.

3 Etat de l'art

3.1 *Clock gating* au niveau RTL

La plupart des efforts d'optimisation de consommation se sont focalisés dans un premier temps sur les niveaux bas du flot de conception à savoir les niveaux qui portent sur les registres, les portes logiques ou encore le *Layout*. Ainsi, des techniques de *clock gating* peuvent être appliquées en observant l'évolution des états des entrées des registres [BBM04] et n'autoriser l'horloge que si des changements d'états sont observés. L'encodage des bus d'adresses dans le but de minimiser les transitions d'états des bits du bus est aussi une autre technique utilisée au niveau logique, citons [YZ04] à titre d'exemple. Ainsi, la technique de *clock gating* est mise en œuvre par différents outils commerciaux tels que *Power Theater* [Seq] et *Power Compiler* [Pow]. *Power Theater* exploite des opportunités permettant d'inhiber l'horloge des registres pour lesquels il n'existe aucun multiplexeur dans leur chemin de rétroaction (*feedback path*). *Power Compiler* utilise la représentation RTL pour appliquer le *clock gating* durant la synthèse. Dans [EB00], les auteurs donnent un aperçu des techniques d'utilisation du *clock gating* et la manière dont les outils industriels comme *Power Compiler* l'appliquent. Ils montrent également les avantages de cette technique au niveau RTL et son impact sur le processus de conception.

De même, la littérature est riche en travaux proposant des méthodologies orientées basse consommation associées à la distribution et la gestion d'horloge dans les systèmes au niveau RTL. Par exemple, une méthodologie basée sur la synthèse comportementale pour construire un arbre d'horloge selon les activités du système est définie dans [FCS+01]. Étant donné une

description d'un préplacement du circuit, l'ensemble des temps d'activité et d'inactivité qui représentent un modèle d'activité pour chaque bloc est construit à partir de la table d'ordonnancement issue de la synthèse et ce pour chaque bloc. L'algorithme de construction de l'arbre d'horloge est une heuristique, basée sur un algorithme récursif de correspondance pondérée, où l'objectif est de regrouper dans le même sous arbre les blocs ayant des motifs d'activité similaires, de sorte que les horloges des blocs peuvent être inhibées au plus proche de la racine et avec une probabilité aussi grande que possible. Dans [GSD99], les auteurs présentent une exploration de l'impact de l'application du *clock gating* sur la construction traditionnelle de l'arbre d'horloge dans le cas de scénarios réalistes.

Par ailleurs, les auteurs de [SS11] étudient les divers procédés de *clock gating* qui peuvent être utilisés pour optimiser la puissance dans les circuits VLSI au niveau RTL ainsi que les divers impacts associés qui résultent de l'application de cette technique d'optimisation de puissance. Les résultats de ces travaux portant sur l'optimisation en puissance consommée d'un arbre d'horloge ont montré que la technique du *clock gating* peut réduire significativement la dissipation de puissance dynamique. Ainsi, ils ont montré l'efficacité de l'exploitation des informations sur les fonctions d'activation d'horloge lors de la génération de l'arbre d'horloge.

Cependant, les approches proposées ne traitent généralement pas des problèmes liés à leur intégration dans les flots de conception existants. Ce point a été abordé dans [DIBM03] en proposant une méthodologie et des algorithmes pour la construction de l'arbre d'horloge qui sont spécifiquement orientée vers l'intégration dans des flots de conception.

Les gains en énergie issus d'une gestion optimisée de l'arbre d'horloge ne sont obtenus qu'en insérant des fonctions logiques dans un circuit décrit au niveau RTL ou logique. Une fois ces fonctions logiques ajoutées, le circuit est par la suite synthétisé, placé et routé correctement par les outils classiques pour obtenir l'implémentation matérielle du circuit. Néanmoins, ces gains importants ne concernent que la puissance dynamique.

3.2 *Power gating* au niveau RTL

Pour réduire la puissance totale d'un système, il est nécessaire de gérer aussi la puissance statique donc la tension d'alimentation. Les éléments qui agissent sur la gestion des tensions d'alimentation et des informations de contrôle associées pour adapter les blocs IP décrits en RTL à un objectif de faible consommation en puissance statique sont nécessairement externes aux outils de synthèse/simulation logique et sont donc définis dans les normes UPF et CPF.

Plusieurs outils industriels utilisent ces normes UPF et/ou CPF dans le développement de leurs solutions pour la modélisation, la conception, la vérification et l'exploration de circuits qui

intègrent les concepts pour la réduction de puissance statique. Par exemple, *Mentor Graphics* fournit le simulateur *Questa Power Aware* [Men] qui se focalise sur la vérification par simulation d'un modèle RTL modifié automatiquement à partir d'un *power intent* décrit en UPF. L'outil *Synopsys Design Compiler* de synthèse permet également d'inclure après la synthèse RTL les cellules orientées gestion de consommation depuis une description d'un *power intent* suivant la norme UPF. [BKS09] et [CC07] développent l'utilisation du standard UPF pour décrire le processus de conception orienté optimisation de puissance en se basant sur la vérification fonctionnelle par simulation au niveau RTL. Afin d'analyser et de vérifier fonctionnellement une spécification d'une architecture faible puissance, *Cadence* a développé l'outil *Conformal Low Power* basé sur CPF qui peut être utilisé tout au long du flot de conception à partir de RTL. Par conséquent, en utilisant les standards UPF et CPF, seules les spécifications fonctionnelles dès le niveau RTL peuvent être considérées pour être associées à des éléments structurels de gestion de tension d'alimentation. Par ailleurs, ces standards comme nous l'avons décrit dans la section 2.6 se focalisent principalement sur la gestion des tensions d'alimentation et ne prennent pas en compte la gestion de la distribution d'horloge, alors que le *clock gating* est reconnu comme étant une technique efficace pour réduire la consommation (Figure II.6).

En conclusion, les approches existantes basées sur une gestion d'horloge opèrent au niveau des blocs IP et ne fournissent pas de solutions génériques et réutilisables au niveau système ce qui est dû à l'absence d'un standard définissant les concepts de la modélisation de la distribution d'horloge. En effet, l'application de ces approches oblige le concepteur à développer et intégrer à chaque système les éléments adéquats à la gestion et à la distribution d'horloge. Ce qui conduit à la fois à des simulations lentes, des difficultés à explorer différentes solutions et des erreurs de conception coûteuses à corriger.

3.3 La réduction de la puissance au niveau transactionnel

Nous présentons dans ce paragraphe une synthèse des travaux de recherche et des méthodologies qui traitent de la modélisation, de la gestion et de l'estimation de la puissance au niveau transactionnel. Dans [NLD05] et [DBD+05], un premier environnement basé SystemC est proposé pour l'estimation de la puissance au niveau PVT. Cet environnement utilise une méthode de construction de modèles de consommation basés sur des transactions. L'idée principale est de construire une structure arborescente hiérarchique qui reprend tous les types et les niveaux de granularité des transferts entre les différents blocs ainsi que les relations d'inclusion possible entre les transactions exprimées à ces différents niveaux de granularité. La valeur de puissance de chaque transaction dans l'arbre peut être fixe ou paramétrable. Ce travail a montré comment

un environnement de simulation basé sur SystemC-TLM peut être amélioré avec des modèles de consommation basés sur les transactions pour l'estimation de puissance.

Par ailleurs, une méthodologie de modélisation de puissance a été considérée dans [BANMD07] pour les composants principaux d'une architecture MPSoC afin d'obtenir une estimation précise de la puissance à des sous-niveaux du niveau PVT. Cette méthodologie suppose que les modèles du niveau PVT et du niveau cycle-précis (CAL) des différents composants sont disponibles et repose sur l'identification des activités pertinentes qui engendrent une consommation importante d'énergie pour chaque modèle PVT et CAL. En effet, cette identification des activités est déduite pour les modèles PVT de celle qui est définie au niveau CAL elle-même caractérisée au niveau porte ou avec des modèles analytiques. L'inconvénient majeur de cette méthodologie est que les modèles CAL des composants d'un SOC ne sont pas toujours disponibles.

Les auteurs de [LV08] proposent une modélisation de la réduction de la consommation de puissance dans des modèles SystemC au niveau transactionnel par des mécanismes de DVFS et de DPM. Cette proposition est l'une des premières qui traite l'application de ces techniques de réduction dans les premières phases du flot de conception, notamment au niveau SystemC-TLM. En considérant pour chaque bloc IP la possibilité d'utiliser des techniques de type DVFS et DPM, les auteurs proposent de décomposer l'état de chaque bloc IP en une phase fonctionnelle et en un mode DPM (par exemple, *On*, *Sleep* et *Off*). La phase fonctionnelle se caractérise par sa durée d'exécution (par exemple en utilisant des fonctions de type *wait()*, *read_compute()*, etc.) et par l'énergie associée qui est issue de mesures effectuées à des niveaux inférieurs au niveau TL. En appliquant le DVFS, la tension et la fréquence sont susceptibles d'être changées indépendamment de la phase fonctionnelle. L'environnement proposé permet d'instrumenter une plateforme SystemC-TLM fonctionnelle existante avec des informations temporelles pour effectuer des estimations de puissance et étudier des politiques de gestion de consommation dans des architectures construites autour de réseaux sur puce (*Network on Chip* - NoC) globalement asynchrone et localement synchrone (GALS). Afin de faciliter la phase d'instrumentation, une bibliothèque *TLM_POWER*, a été développée. Elle est composée d'un ensemble de classes générique C++ qui modélisent les différentes phases fonctionnelles combinées avec les modes de DPM. Cet environnement a été utilisé afin d'évaluer par exemple les mécanismes de gestion d'énergie conçus particulièrement pour la plate-forme Magali du CEA [Leb09]. En dépit de l'optimisation et des gains énergétiques possibles issus de l'utilisation de cette première version, la plateforme souffrait de certaines limitations. Nous citons en particulier les limitations suivantes :

– Le concepteur doit définir des classes intermédiaires et des types énumérés qui ne peuvent représenter que des valeurs discrètes de puissance d'un module SystemC.

– La consommation d'énergie d'un module n'intègre pas la consommation de ses sous-modules.
– Les valeurs de la puissance et de l'énergie sont définies par des valeurs brutes qui peuvent conduire à des erreurs du fait de l'absence de contrôle sur les unités utilisées (par exemple, milli-watts et pico-watt).

Les défauts de la première version de *TLMPOWER* ont été surmontées dans une seconde version nommée *TLMPOWER2* [Mat]. En effet, la bibliothèque *TLMPOWER2* introduit des classes de base pour la définition de la consommation de puissance et d'énergie ainsi que le mode de fonctionnement d'un module au cours d'une simulation y compris les modules hiérarchiques. Elle définit aussi deux classes pour spécifier les unités physiques de puissance et d'énergie. Tous les opérateurs arithmétiques standards sont surchargés pour avoir le comportement attendu. Étant donné que *TLMPOWER2* est basé sur l'approche mode/phase, les appels TLM ne sont pas annotés, il est nécessaire d'expliciter à l'ordonnanceur SystemC la quantité de temps où le module reste dans le mode/phase présent pour effectuer les calculs de puissance et d'énergie. Ainsi, cette bibliothèque utilise seulement le style de codage AT (section 2.3). De plus, elle n'intègre pas la modélisation des sockets TLM. Une troisième version de la bibliothèque *TLMPOWER3* [GY12] a été produite pour permettre à la fois l'approche mode/phase et l'approche energy-per-transaction. Cette dernière est mise en place par une extension de la classe *Generic payload*. En effet, *TLMPOWER3* traite la modélisation transactionnelle avec le style de codage TL (voir section 2.3). Outre les unités physiques proposées dans *TLMPOWER2*, *TLMPOWER3* ajoute d'autres unités pour la tension, les dimensions et la surface avec les opérateurs surchargés de manière appropriée afin de rendre les estimations plus précises. Par conséquent, ces bibliothèques permettent de modéliser et d'évaluer une stratégie de gestion de consommation appliquée à un modèle fonctionnel d'une architecture décrite en SystemC-TLM. Cette approche nécessite donc qu'un code spécifique SystemC relatif à la stratégie de gestion de consommation soit inséré dans tous les modules SystemC fonctionnels afin d'instrumenter le modèle avec des informations et des changements d'états qui reflètent la stratégie de gestion de la consommation d'énergie et de puissance. Durant la simulation, ces changements d'états permettent de calculer la consommation d'énergie des blocs IP mais ils ne modélisent pas réellement les contrôles émis par la stratégie de gestion de consommation qui impacte les performances et le comportement fonctionnel des blocs IP.

Une autre approche d'instrumentation orientée puissance consommée pour des modèles TLM est proposée dans [BMM13]. Les auteurs se focalisent essentiellement sur les modèles de distribution de trafic au niveau TLM dans un but d'obtenir des estimations de température correctes via un modèle thermique activé suivant une granularité adaptée. Leur idée consiste

à répartir de façon pertinente la consommation de la puissance associée à une fonctionnalité donnée (les accès à la mémoire) sur toute la période de simulation. Cette méthodologie vise ainsi à analyser l'exactitude et l'efficacité d'une stratégie de gestion de puissance et de ses effets thermiques.

Cependant, ces approches [BMM13], [Leb09] et [LV08] ne s'appuient pas sur les concepts des normes UPF et CPF. Ainsi, la mise en œuvre et la vérification au niveau RTL d'un modèle décrit au niveau TLM doit être entièrement reprise manuellement ce qui peut provoquer fatalement des erreurs. De plus, ces approches ne considèrent pas le principe de séparation entre les aspects fonctionnels et les aspects non-fonctionnels ce qui limite leurs capacités pour supporter l'exploration du *power intent*.

Le principe de séparation des aspects fonctionnel et consommation est considéré dans [GHF+14] en mettant l'accent sur la modélisation de la stratégie de la gestion d'alimentation. En effet, les états de consommation d'un composant sont décrits suivant une PSM (*Power State Machine*) et sont ensuite rattachés directement à des événements fonctionnels générés en simulation par le modèle fonctionnel. Une telle approche ne facilite pas réellement la vérification de la stratégie de gestion de consommation en relation avec le comportement fonctionnel du système.

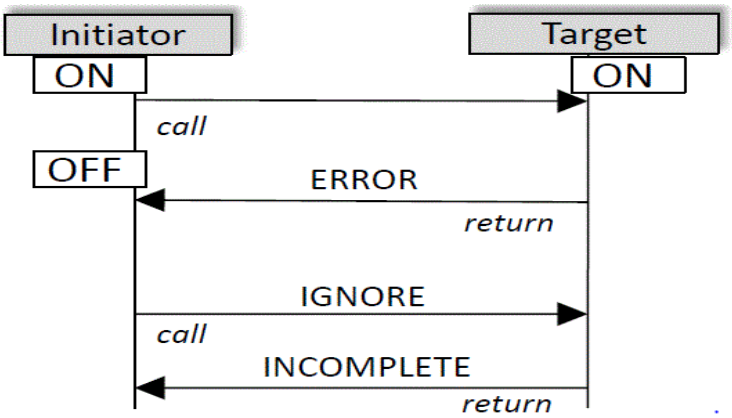


Figure II.12 – Modification des *datapaths* TLM en intégrant les états de puissance

En revanche, les auteurs de [MM14] s'intéressent à cette vérification du *power intent* par la surveillance des transactions TLM échangées entre les PD eux-mêmes définis à partir d'une expression de type UPF. Ce travail est l'extension de [MM13] dans lequel une approche pour instrumenter un *power/clock intent* dans un module SystemC-TLM d'un système est introduite. Dans cette approche, comme indiqué sur la Figure II.12, les transactions fonctionnelles du standard TLM sont étendues pour notifier les transitions d'états avec des attributs permet-

tant d'indiquer si les métadonnées des transactions issues/reçues par les blocs sont valides ou non (le bloc est dans un état OFF comme sur l'exemple de la Figure II.12).

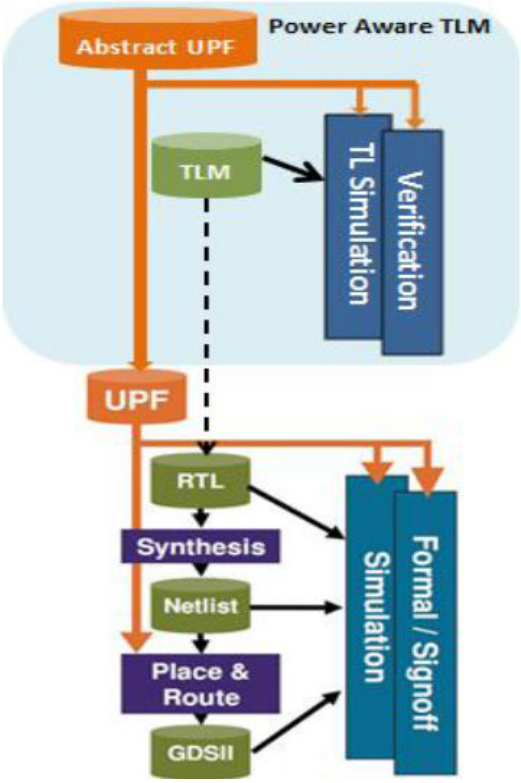


Figure II.13 – Extension du flot de conception *power* vers le niveau TLM [MPA11]

Outre les travaux mentionnés ci-dessus, l'environnement *PwARCH* [MPA11] permet l'exploration de l'espace de conception de différentes architectures de gestion de consommation au niveau transactionnel. Ceci est réalisé en augmentant un modèle SystemC-TLM d'une architecture fonctionnelle avec une abstraction des concepts UPF nécessaires à la simulation. En effet, cet environnement aide à construire un modèle abstrait d'une structure de gestion de consommation (appelée architecture *power*) selon les principes issus de deux techniques, le *power gating* et une alimentation de type multi-tension. Cette abstraction permet de valider au niveau TLM un *power intent* en vérifiant en particulier la cohérence entre les états fonctionnels de l'application exécutée et ceux de l'architecture *power*. Pour cela, l'approche proposée applique une séparation des concepts fonctionnels et des concepts relatifs à la gestion de la puissance qui est analogue à celle définie par UPF (section 2.6). Cette approche permet également d'évaluer, localement et globalement, la consommation d'énergie et de puissance pour

le *power intent* considéré. Par conséquent, comme l'illustre la Figure II.13, la connexion avec le flot de conception RTL est réalisée à travers la génération automatique d'un fichier UPF correspondant à la description abstraite de l'architecture *power* définie au niveau TLM.

Afin d'atteindre ces objectifs, une méthodologie est proposée en se basant sur l'environnement *PwARCH*. La Figure II.14 présente le flot global de la méthodologie. Cette méthodologie se compose de trois étapes séquentielles et d'une étape orthogonale dédiée à la vérification. L'approche proposée permet d'ajouter de manière structurée le *power intent* ainsi que les stratégies de gestion de la consommation à un modèle fonctionnel de niveau TLM existant. La méthodologie est itérative, permettant ainsi l'exploration de l'espace de conception du *power intent*. Dans la suite, nous décrivons brièvement les différentes étapes de cette méthodologie.

La spécification du *Power intent* :

Lors de cette étape, tous les éléments nécessaires à la spécification du *power intent* sont définis. Il s'agit de superposer aux composants fonctionnels existants les éléments de l'architecture *power* afin de simuler par la suite leurs impacts à la fois sur la puissance dissipée et sur la capacité du composant à exécuter un comportement fonctionnel. Pour cela, les concepts définis par le standard UPF ont été abstraits au niveau TLM. La hiérarchie de composition définie dans la norme UPF est modélisée au niveau TLM avec la même sémantique qu'UPF (voir section 2.6). Le comportement de cette architecture *power* et son contrôle (la stratégie de gestion de consommation) doivent également être définis lors de cette étape. Dans cette perspective, de nouveaux concepts ont également été ajoutés pour définir les notions de *Design Element* (DE), d'observateurs, de *power monitor* et de *Domain Power Controller* (DPC). Chaque module de l'architecture fonctionnelle SystemC-TLM est ainsi attaché à un objet de type DE qui représente le module dans le *power intent*. Ceci renforce la séparation entre le modèle fonctionnel et le modèle orienté *power*. Ainsi, chaque PD est défini par un ensemble d'objets DE qui partagent les mêmes *supply nets*. La valeur de la tension d'alimentation d'un PD est définie par un *supply net* associé à travers d'un *power switch* (un par PD) qui peut être ouvert ou fermé. Les concepts d'observateur et de *power monitor* sont utilisés pendant les étapes de vérification et d'estimation de la consommation de puissance ou d'énergie. Les *power monitors* servent en effet à collecter les informations issues des observateurs. Les DPC, un par PD, sont responsables de changer l'état des *power switchs*. Tous ces concepts et les classes associées font l'objet de la librairie *PwARCH*.

Modélisation de l'unité de gestion de la puissance consommée (PMU) :

Pour contrôler les états d'alimentation locaux et globaux, une unité de gestion de puissance (*Power Management Unit* - PMU) est modélisée comme un module SystemC-TLM supplémentaire communiquant par des transactions sur le bus avec les autres modules de la plateforme

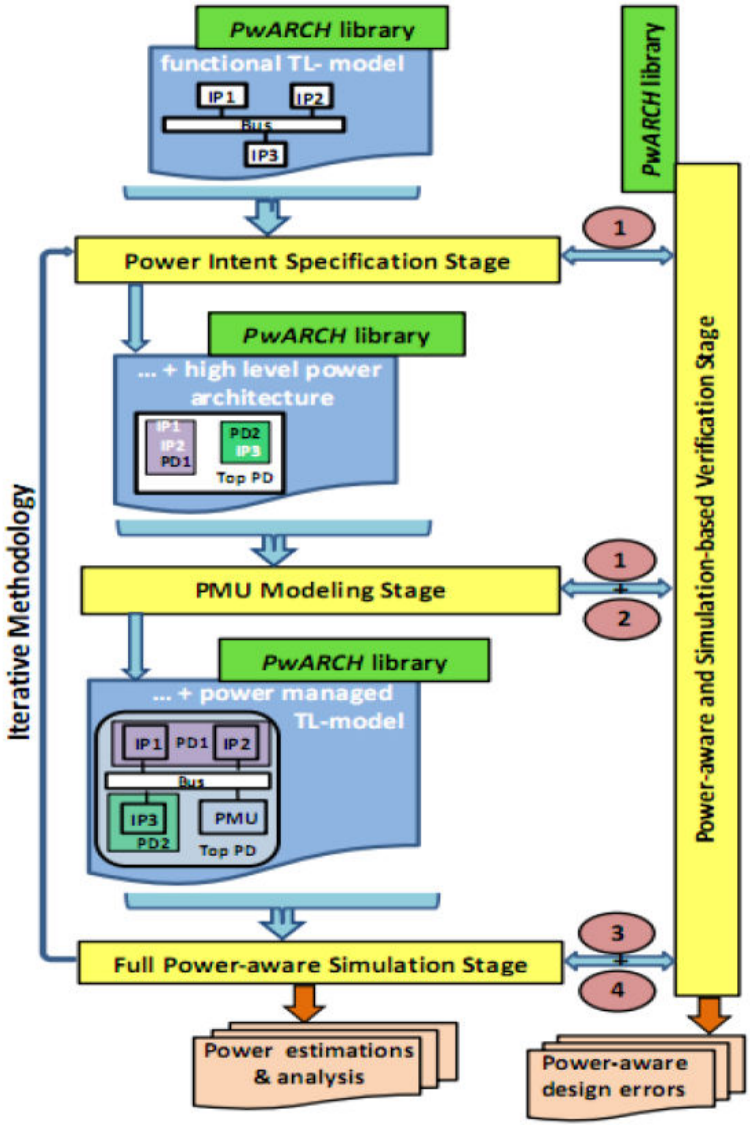


Figure II.14 – Flot de la méthodologie basée sur les *Power Domains* [MPA11]

pour assurer l'interface avec le modèle fonctionnel en vue de fixer les états des PD en relation avec le niveau de performance souhaité tout en minimisant la puissance consommée. Cette unité est composée par un module *Power Manger* (PM) et les DPC. Le rôle du PM consiste à implémenter une stratégie de gestion des PD. Cette stratégie est basée sur une PST (Power State Table) afin de contrôler les différents PD en changeant l'état des *supply nets* ou des *power switchs*. Un DPC est en charge d'ajuster selon une séquence bien déterminée l'état des éléments d'alimentation de son PD. Il reçoit les ordres de transitions d'états à travers des signaux issus

du PM. Le PMU est la seule entité SystemC-TLM ajoutée au modèle fonctionnel dans l'approche proposée. La dernière étape de la méthodologie consiste à simuler le comportement du système obtenu : modèle fonctionnel et architecture *power* couplés.

Simulation du système complet :

La simulation permet de vérifier la cohérence des états fonctionnels avec les états induits par le modèle du *power intent*. En effet, le comportement du système est considéré comme cohérent si aucune violation de propriétés telles que celles décrites dans le paragraphe suivant n'est détectée durant la simulation. Des fichiers de traces des valeurs successives de consommation de puissance sont également générés lors de la simulation. Ces fichiers permettent d'analyser et d'effectuer des comparaisons entre différentes alternatives de *power intent*.

Vérification des propriétés *power-aware* en simulation :

L'objectif de cette étape orthogonale aux précédentes est de vérifier des propriétés orientées consommation de puissance après chacune des étapes précédentes en utilisant une approche par contrats. Pour ce faire, des contrats génériques de type assertion sont ajoutés à *PwARCH* en particulier dans les objets de type *Observer* et *Monitor*. Pour compléter la vérification, d'autres contrats (assertions) peuvent être introduits dans le code fonctionnel qui déclencheront une exception lors de la violation d'un de ces contrats. Par exemple, une assertion peut consister à tester si une transaction est reçue par un bloc fonctionnel alors que ce bloc est dans un état OFF et qu'il n'a pas la capacité de déclencher son propre réveil sur réception d'une transaction. Dans ce cas une exception est levée.

Pour récapituler, l'environnement *PwARCH* permet de gérer et d'évaluer la consommation de puissance et d'énergie durant une simulation d'un modèle fonctionnel SystemC-TLM. Il se focalise essentiellement sur la modélisation et la gestion induite par un *power intent* correspondant à une abstraction d'une description UPF et donc des concepts inclus dans cette norme. Cependant, *PwARCH* ne s'intéresse qu'à la modélisation des variations des tensions d'alimentation par *power gating* et l'ajustement de ces tensions mais ne prend pas en compte toutes les autres techniques qui peuvent agir sur la consommation de puissance (variation de fréquence, DVFS, *clock gating*, AVS, etc.).

Outre les travaux et les environnements déjà présentés, un ensemble d'approches et d'outils ont été proposés afin de faciliter l'estimation et l'analyse de la consommation de puissance et d'énergie au niveau transactionnel. Nous citons par exemple les outils suivants :

– L'environnement de simulation Pktool [VC09] est un outil d'estimation de l'énergie dédié pour les systèmes décrits en SystemC-TLM. Cet environnement s'exécute simultanément

avec la simulation SystemC native. Il travaille en arrière-plan de manière transparente et sans affecter le comportement du noyau SystemC. A travers des macros C++, cet outil surveille les appels aux fonctions des transactions afin de mettre à jour lors de la simulation les contributions de dissipation de chaque module SystemC-TLM. Il permet d'associer à chaque module un ensemble de modèles de consommation qui sont liées à chaque fonction associée à une transaction. Pktool n'estime que les coûts énergétiques liés aux transactions qui sont dérivés par une approche de macro-modélisation basée sur des évaluations obtenues à bas niveau, par exemple au niveau porte logique.

– De côté industriel, Synopsys a proposé l'outil *Innovator Virtual Prototype* dont l'objectif est d'accélérer la création des modèles au niveau transactionnel. Cet outil permet de créer des nouveaux blocs IPs avec des interfaces permettant de décrire des horloges, des tensions et des états d'alimentation. Les paramètres relatifs à la puissance d'un bloc IP sont insérés par l'utilisateur avant la simulation en utilisant l'interface graphique. Ces paramètres sont calculés à l'aide des outils du niveau inférieur comme *Power Compiler*. Récemment, *Innovator* a été remplacé par l'outil *Platform Architect* (PA-MCO) [Syn]. PA-MCO est un environnement d'analyse de performance basé sur SystemC-TLM mais qui a été étendu pour permettre l'analyse de la puissance consommée. Cette analyse de la consommation d'énergie est effectuée au cours d'une simulation d'une plateforme virtuelle TLM avec des moniteurs affectés à chaque composant. L'analyse de la puissance est effectuée sans interférer ou modifier les blocs IP TLM. En effet, la définition des états d'alimentation et de la consommation d'énergie par l'intermédiaire de PSM se fait à l'aide d'une interface de script TCL. Les changements de ces états peuvent être déclenchés à partir du modèle TLM à travers différents événements comme des changements des états des signaux ou des appels à des fonctions du logiciel embarqué. La liaison entre les états d'alimentation et les déclencheurs est également définie en utilisant la même interface de script TCL. Lors d'une simulation, toutes les données tracées telles que les états d'alimentation, l'évolution de la consommation de la puissance ainsi que les valeurs des fréquences et des tensions sont recueillies dans une base de données qui est ensuite exploitée pour l'affichage des résultats. Ces résultats d'analyse de puissance dépendent essentiellement des PSM spécifiés par le concepteur. Toutefois, définir cette dépendance mutuelle entre les PSM et le comportement fonctionnel peut ne pas être trivial en utilisant PA-MCO. En effet, la stratégie de gestion de puissance n'est pas explicitée par cette méthode et ainsi il peut ne pas être simple d'associer de manière cohérente par *Power Domain* et par *Clock Domain* des valeurs de fréquence et de tension suivant les états ou événements

fonctionnels observés.

– *Docea Power* a développé l'outil *Aceplorer* [Ace] qui a pour but de répondre aux besoins de modéliser et de simuler la consommation d'énergie et les comportements thermiques de systèmes électroniques. *Aceplorer* permet la modélisation de systèmes hétérogènes, avec des blocs mixtes numériques analogiques et avec différents niveaux de complexité des modèles. Pour obtenir une description de l'évolution de la puissance dissipée et du comportement thermique, un scénario fonctionnel doit être décrit. Ces scénarios peuvent être fournis par l'intermédiaire d'une simulation d'un modèle SystemC- TLM fonctionnel. Cependant, avec ce type d'approche, un effort non négligeable est nécessaire pour établir par du code intrusif dans chaque bloc SystemC-TLM la relation et la synchronisation entre le modèle fonctionnel SystemC-TLM et les états décrivant la puissance dissipée (par PSM).

4 Conclusion

Ce chapitre nous a permis de présenter un certain nombre de techniques et de standards de gestion et d'estimation de la consommation d'énergie aux différents niveaux de conception d'un système.

Malgré l'importante littérature sur le sujet de la réduction de la consommation d'énergie, il apparaît que de nombreuses avancées sont encore nécessaires pour faire face aux défis liés aux évolutions de la technologie. En effet, notre analyse de certaines approches montre que celles-ci sont principalement restreintes soit à l'estimation de la consommation par l'insertion de code intrusif dans le code fonctionnel afin de mimer la stratégie de gestion de consommation soit à la définition et la gestion de la tension d'alimentation. Ces approches ne décrivent généralement pas les mécanismes permettant de construire et de gérer un arbre d'horloge au niveau système. Cela est dû essentiellement à l'absence d'un standard décrivant la structuration d'un arbre d'horloge.

Dans cette optique, nous proposons dans le chapitre suivant un environnement *ClkARCH* qui permet de spécifier la structuration d'un arbre d'horloge dans une architecture décrite en SystemC-TLM afin de supporter une stratégie de gestion de *Clock Domains*.

Chapitre III

Méthodologie d'insertion de stratégies de gestion d'horloge au niveau transactionnel pour les systèmes sur puce

1 Introduction

Dans ce chapitre, nous présentons notre environnement *ClkARCH* permettant de modéliser un arbre d'horloge au niveau SystemC-TLM afin de gérer et d'estimer la consommation de la puissance dynamique au niveau transactionnel. Dans nos travaux, nous ciblons des plateformes virtuelles éventuellement capables d'exécuter le code logiciel applicatif y compris le *middleware*, l'SE et les pilotes. Ainsi, nous visons à définir des modèles qui agissent à la fois sur le *power* et sur la performance et qui permettent de structurer une architecture décrite au niveau transactionnel, que cette architecture soit un prototype virtuel (niveau *Programmer's View*) ou décrite au niveau algorithmique. Étant donné l'absence de standardisation de la description des mécanismes de création et de gestion d'un arbre d'horloge dans les normes existantes relatives à la gestion de puissance comme UPF et CPF, nous avons analysé deux architectures issues de deux différents constructeurs : OMAP4 de *Texas Instruments* [OMAb] et SPEAr1340 de *STMicroelectronics* [ST]. L'objectif est d'identifier les principes qui peuvent s'appliquer dans une approche générique de modélisation d'arbre d'horloge en SystemC-TLM. Dans la suite, nous présentons d'abord les résultats de cette analyse en déduisant les principes à retenir pour une modélisation en SystemC-TLM d'un arbre d'horloge. Ensuite, nous décrivons la structure générale de l'environnement *ClkARCH* ainsi que le flot de notre méthodologie. Enfin, le dernier paragraphe décrit les cas d'utilisation que nous avons menés pour évaluer les performances des implémentations réalisées.

2 Concepts pour construire un arbre d'horloge

Notre analyse des deux architectures OMAP4 et SPEAr1340 se focalise principalement sur la partie de gestion de puissance de manière à en déduire les principes qui peuvent servir à définir un modèle au niveau transactionnel en adéquation soit avec une phase d'exploration soit avec une phase d'estimation plus précise et de validation du logiciel applicatif vis-à-vis de la stratégie de gestion de puissance. Ces principes sont alors confrontés aux besoins de nos travaux. Le choix d'étudier ces deux architectures a été guidé par le fait qu'elles utilisent des techniques avancées pour optimiser la consommation de puissance mais elles sont assez différentes dans leur approche de contrôle de cette consommation. L'OMAP4 utilise un contrôle très fortement orienté matériel alors que le SPEAr1340 fait intervenir le logiciel de manière plus présente dans ce contrôle.

2.1 Analyse de l'architecture OMAP4470

L'architecture OMAP4470 fait partie de la famille OMAP4 développée par *Texas Instruments* (TI). Cette plateforme a été développée pour équiper des smartphones ou des équipements dédiés à l'Internet mobile. Un des objectifs principaux qui ont guidé sa conception et son développement est de définir une architecture permettant d'atteindre des performances élevées avec une consommation de puissance réduite.

Ce type d'architecture contient à la fois des unités de traitement, de stockage et de communication adaptées vis-à-vis des fonctionnalités/applications potentiellement intégrées dans les équipements cibles et de nombreux mécanismes utilisables pour réduire la consommation de puissance ou d'énergie. Le synoptique fonctionnel de cette architecture est illustré dans la Figure III.1. L'OMAP4 est développé en technologie 45nm et contient plusieurs blocs. D'un point de vue logiciel, l'OMAP4 supporte les OS et RTOS Microsoft Windows Mobile, Symbian, Linux avec Android ou Limo et intègre différentes fonctions de communication comme GPS, WiFi, FM, etc. La gestion de la consommation de puissance et d'énergie s'effectue dans ce circuit à plusieurs niveaux avec pour chaque niveau des mécanismes soient logiciels soient matériels.

L'architecture matérielle est structurée autour d'une hiérarchie de bus d'interconnexion qui regroupe fonctionnellement les différentes unités du système. Comme illustré dans la Figure III.2, des unités particulières implémentent les mécanismes de gestion de puissance et de gestion de l'arbre d'horloge de l'OMAP4470. Il s'agit des unités suivantes :

– Le PRCM (*Power, Reset and Clock Management*) qui lui-même contient un PRM (*Power and Reset Management*) et deux *Clock Managers* (CM1 et CM2). Ces trois modules sont

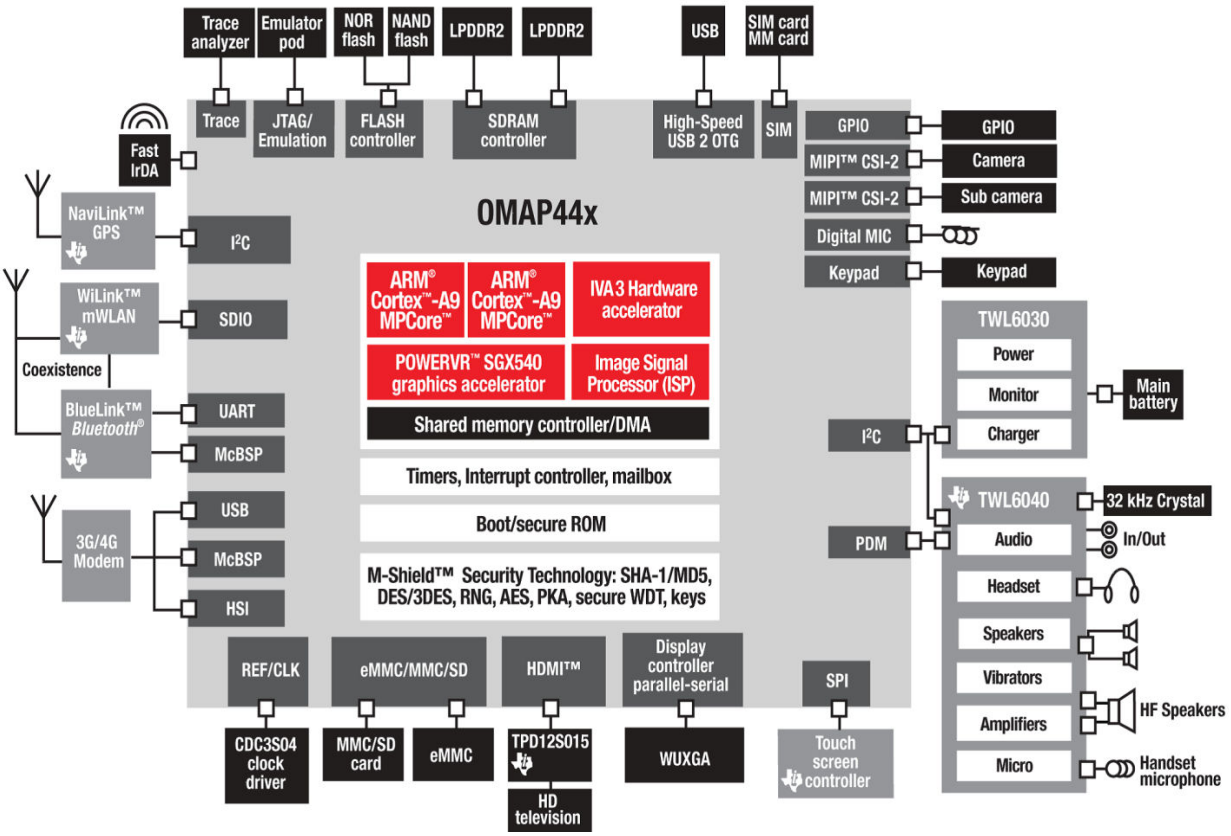


Figure III.1 – Synoptique de l'architecture de l'OMAP4470

constitués de registres, de diviseurs, de multiplexeurs et de portes logiques.

- Le SCRM dont le rôle est de gérer au niveau système les horloges et les signaux de reset.
- Des unités SR_MPU, SR_IVA et SR_CORE dont la fonction est de gérer l'AVS de sous-systèmes spécifiques.

L'ensemble des informations sur le PRCM et le SCRM sont disponibles dans le *Technical Reference Manual* [OMAb] de TI. A noter que le troisième chapitre du manuel décrivant ces deux modules contient 769 pages ce qui reflète la complexité de gestion de la puissance dans un tel circuit. Dans le but d'intégrer plusieurs techniques de *power management* dans le circuit, TI a structuré l'architecture de l'OMAP4470 en trois niveaux de gestion de ressources : le niveau *clock*, le niveau *power* et le niveau *voltage*. Chaque niveau possède une politique de gestion, l'ensemble est regroupé dans le PRCM. Dans la suite, nous nous intéressons plus particulièrement au niveau *clock*.

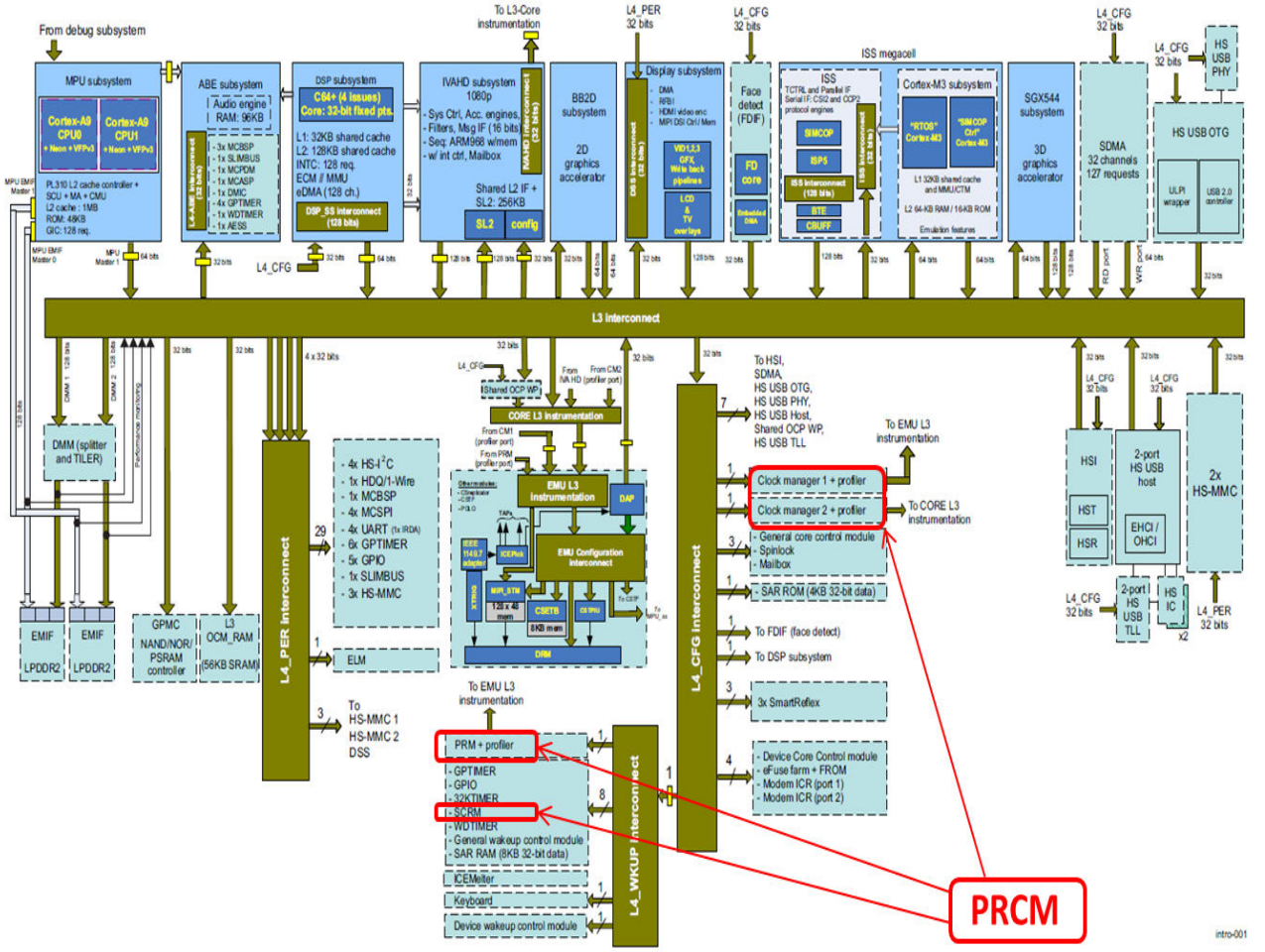


Figure III.2 – Synoptique complet de l'OMAP4470

2.1.1 Les *Clock Domains* et leur gestion dans l'OMAP4470

La définition d'un *Clock Domain* (CD) donnée par TI pour ce circuit est la suivante : «Un CD est un groupe de blocs IP contrôlés par un même Clock Manager (CM) localisé dans le PRCM» (voir Figure III.3). Ainsi, il est possible d'appliquer la technique de *clock gating* au sein de chaque CD. Chaque CD est inclus dans un PD. Dans l'OMAP4470, les horloges appliquées à un bloc IP peuvent être de deux catégories : des horloges fonctionnelles et des horloges d'interface (voir Figure III.4). Les horloges fonctionnelles sont celles qui permettent au bloc IP d'exécuter sa fonction logique alors que les horloges d'interfaces sont utilisées dans les opérations de communication sur le ou les bus sur lesquels le bloc est connecté. La gestion des horloges d'interface est effectuée directement au niveau du module lui-même.

L'architecture générale et simplifiée de l'arbre d'horloge composée des modules PRM, CM1,

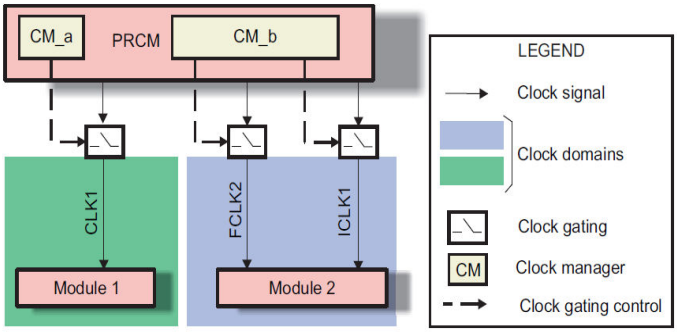


Figure III.3 – Exemple de structuration d'une architecture en *Clock Domains*

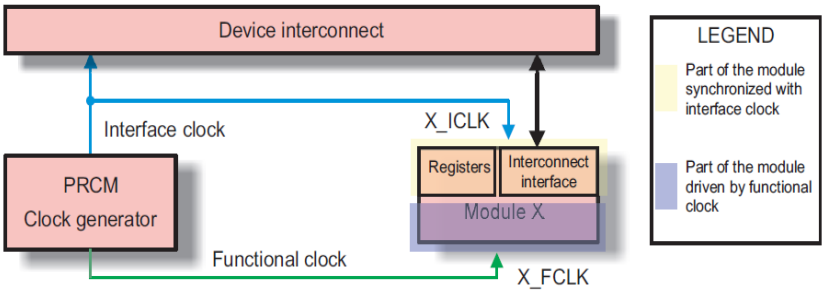


Figure III.4 – Horloges fonctionnelles et d'interface des blocs IP de l'OMAP4470

CM2 et des DPLL (*Digital Phase Locked Loop*) est illustrée sur la Figure III.5. Le bloc SCRM reçoit une horloge externe à 32 kHz (SYS_32K) et une horloge stabilisée externe (SYS_CLK) également à 32 kHz. L'horloge SYS_32K est utilisée pour alimenter les logiques de réveil *Wake-up* des blocs IP qui possèdent cette capacité. Ces deux horloges sont transmises au PRM. Le PRM contrôle les sources d'horloge des DPLL qui à leur tour fournissent les signaux d'horloge pour les CM1 et/ou CM2. Le CM1 ou le PRM contrôle et produit également les signaux d'horloge pour les DPLLs IVA et MPU. Les modules CM1 et CM2 contrôlent et produisent les signaux d'horloge vers tous les autres blocs IP de l'architecture.

L'architecture de l'OMAP4470 est partitionnée en 21 *Clock Domains*. Chaque CD peut contenir plusieurs blocs. Nous n'énumérons pas ici tous les CD et leurs compositions, nous citons par exemple le domaine CD_CAM qui contient les blocs IP FACE_DETECT et ISS.

2.1.2 Les états d'un *Clock Domain* et les transitions entre états

Un CD peut être dans l'un des états suivants : *ACTIVE*, *IDLE_TRANSITION* ou *INACTIVE* (Figure III.6). Dans l'état *ACTIVE*, toutes les horloges à destination de tous les blocs

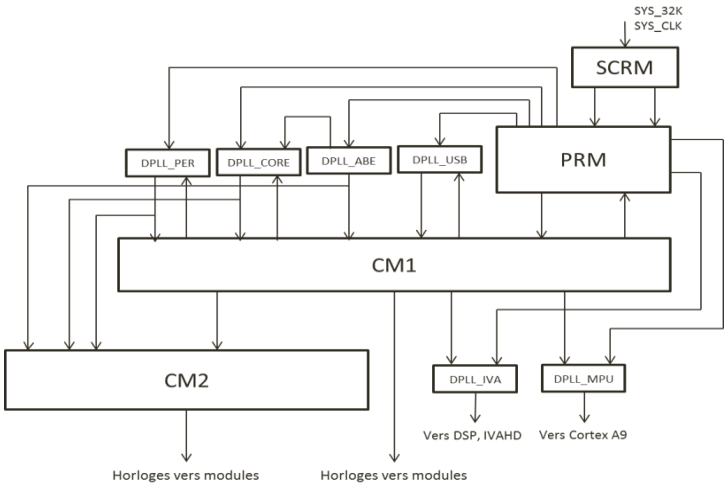


Figure III.5 – Distribution d'horloge dans l'OMAP4470

initiateurs et des blocs cibles ayant un état différent de *IDLE*, contenus dans un CD, sont actives (horloges d'interface et horloges fonctionnelles). Dans le cas de l'état *IDLE_TRANSITION*, tous les initiateurs du CD sont en état *STANDBY*, les horloges fonctionnelles vers les blocs cibles actifs sont actives. En mode *INACTIVE* tous les blocs cibles sont dans l'état *IDLE*, tous les initiateurs sont dans l'état *STANDBY*. Toutes les horloges sont alors inhibées. Pour faire transiter le CD de l'état INACTIVE à l'état *ACTIVE*, un événement de réveil (*Wake-up*) doit être déclenché. Le changement d'état est contrôlé par le PRCM qui transmet des signaux de contrôle au CM. Ces signaux correspondent à des valeurs bien déterminées, par exemple, NO_SLEEP force le CD à rester à l'état *ACTIVE*.

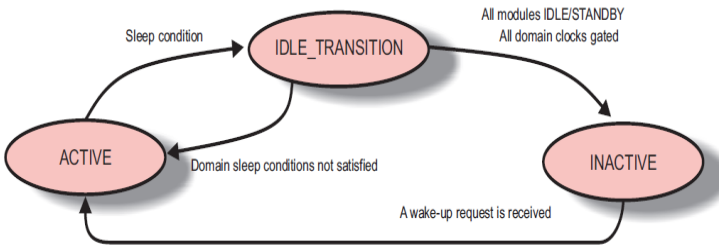


Figure III.6 – États d'un *Clock Domain*

L'OMAP4470 utilise une notion de dépendance pour valider ou non la transition en mode *INACTIVE* d'un CD. Trois types de dépendances sont implémentés : des dépendances statiques, des dépendances dynamiques et des dépendances de réveil (*wake-up*). Ces dépendances sont programmées dans les registres des CM (registres placés dans l'espace d'adressage) et testées

par ceux-ci pour valider ou non le changement d'état.

2.1.3 Le contrôle du changement d'état des *Clock Domains*

Le PRCM contrôle les changements des états des CD en fonction des requêtes transmises par les blocs initiateurs de l'architecture. Un bloc initiateur peut demander au PRCM de ne jamais être en mode *clock gated* ou au contraire le forcer à maintenir le mode *clock gated*. Il peut également demander à ce que le PRCM active certaines horloges (fonctionnelles et/ou d'interface) pour initier une transaction sur l'interconnexion. Enfin, un bloc initiateur en *clock gated* peut recevoir une transaction depuis le bus sur lequel il est connecté, dans ce cas une transition de réveil est initiée qui peut conduire à un changement d'état du CD. Suivant l'état du bloc (le bloc IP retourne à son état *ACTIVE* ou *STANDBY*), la décision d'appliquer le *clock gating* est prise localement par le PRCM. Avec les blocs cibles le protocole est similaire, pour une cible, l'état est appelé *IDLE* et non pas *STANDBY*. Un bloc cible peut être doté d'une capacité de réveil ou non. La capacité de réveil d'un bloc cible lui permet de changer d'état sans nécessiter de décision initiale venant du PRCM. Ainsi, un bloc cible avec la capacité de réveil lorsqu'il est en mode *IDLE* peut générer un événement de réveil vers le PRCM qui en retour active l'horloge ou les horloges nécessaires au bloc. Cet événement peut être la conséquence d'une requête d'un DMA par exemple. La décision du *clock gating* sur un CD est prise par le PRCM (CM1 ou CM2 essentiellement) si l'ensemble des blocs alimentés par l'horloge considérée est soit en mode *STANDBY*(initiateurs), soit en mode IDLE (cibles).

2.1.4 Les sources des horloges : les *Digital Phase Locked Loop* - DPLL

Les DPLL sont les principales sources des signaux d'horloge à l'intérieur du circuit. L'architecture fonctionnelle d'une DPLL donnée dans [OMAb] est illustrée sur la Figure III.7. Il s'agit d'un modèle classique de DPLL.

Le générateur d'horloge utilise la fréquence de référence REF_CLK, éventuellement externe au circuit, pour synthétiser une horloge F_{dpll} à partir de laquelle différentes horloges sont générées au travers des diviseurs. À la sortie du générateur d'horloge la fréquence obtenue du signal F_{dpll} est égal à

$$F_{dpll} = \frac{REF\_CLK * 2 * M}{N + 1} \tag{III.1}$$

où M et N sont des facteurs des multiplication et division contenus dans des registres. À partir de ce signal, d'autres facteurs de division peuvent être appliqués ($M2$, $M3$, $M4$, $M7$). Tous les facteurs de division et de multiplication sont programmables. Sur chaque horloge de sortie, le

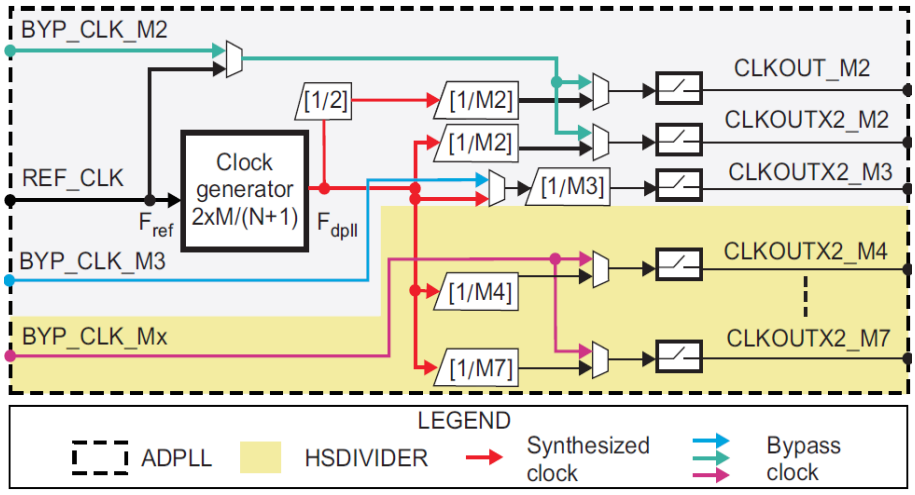


Figure III.7 – Architecture générale d'une DPLL dans l'OMAP4470

clock gating peut être appliqué. Enfin, des multiplexeurs permettent de sélectionner différentes valeurs de facteurs de division ou d'utiliser une horloge de dérivation (horloge de *bypass*). Pendant que les DPLL basculent vers un autre mode de puissance le système est en mode de dérivation (alimenté par l'horloge *bypass*). Cette horloge est utilisée aussi au moment de la réinitialisation du système (RESET).

Ainsi, l'architecture de l'OMAP4470 révèle de nombreux mécanismes réalisés en matériel permettant d'ajuster au plus près l'état de consommation de puissance en fonction de l'activité fonctionnelle des blocs. La difficulté réside alors dans la définition et la validation du contrôle des états de puissance qui doivent être cohérents avec les états fonctionnels.

2.2 Analyse de l'architecture du SPEAr1340

Le circuit SPEAr1340 [ST] est un système sur puce de *STMicroelectronics* de la famille des microprocesseurs embarqués SPEAr ® (*Structured Processor Enhanced Architecture*). Ce produit est conçu pour des applications grand public et professionnelles qui impliquent des interfaces homme-machine évoluées et des hautes performances de calcul telles que les tablettes à faible coût, les téléphones multimédias ou des panneaux intelligents. Cette architecture supporte *Linux*, *Android* ou *Windows Embedded Compact 7*.

Globalement, cette architecture est moins complexe du point de vue contrôle que l'OMAP4470 car elle vise des produits intégrant un nombre de fonctionnalités plus limité par rapport à un smartphone récent comme le montre le diagramme fonctionnel de ce circuit illustré sur la Figure III.8. Cependant, comme cette architecture est conçue pour des systèmes pouvant

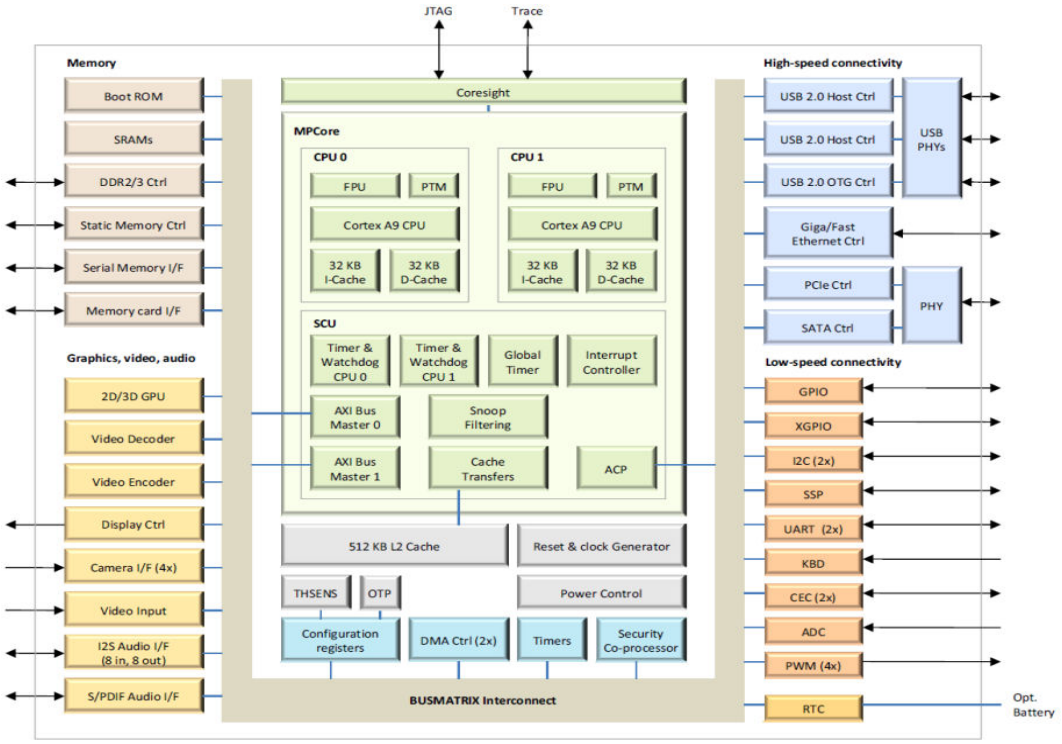


Figure III.8 – Architecture fonctionnelle du SPEAr1340

être embarqués avec des performances de calcul et graphiques élevées, elle intègre d'une part des capacités importantes de communication et d'autre part des mécanismes de réduction de consommation de puissance et d'énergie. Comme dans le cas de l'architecture OMAP4470, nous présentons les mécanismes retenus pour le contrôle de la consommation de puissance au niveau *clock* dans cette architecture SPEAr1340.

2.2.1 Les *Clock Domains* du SPEAr1340

Le SPEAr1340 contient comme illustré à la Figure III.8 un bloc appelé RCG (*Reset Clock Generator*) dont la fonction est de fournir les signaux d'horloge et de *reset* aux autres blocs. Le bloc RCG est localisé dans le PD *Always-On* du circuit. À partir de signaux issus d'oscillateurs externes, trois PLL situées dans le RCG permettent de produire l'essentiel des signaux d'horloge nécessaires (Figure III.9). À la sortie des PLL, un bloc *Clock Sys* est utilisé pour sélectionner le signal d'horloge qui est ensuite appliqué aux autres blocs du système. Ce bloc est essentiellement constitué de fonctions de multiplexage pour effectuer cette sélection. En aval de ce dernier bloc, un autre bloc *Gate Unit* permet de réaliser la fonction de *clock gating* sur les horloges

fournies par le bloc précédent. Ainsi, nous remarquons de fortes similitudes entre les deux solutions de distribution des signaux d'horloge adoptées dans les deux architectures OMAP4470 et SPEAr1340.

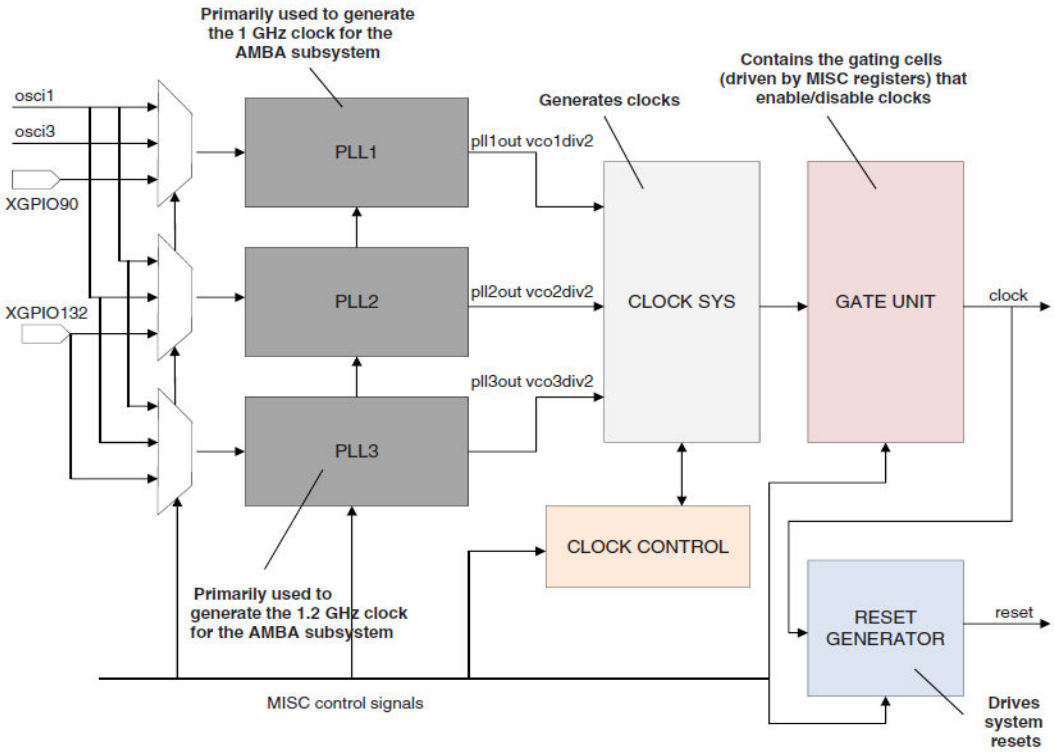


Figure III.9 – Architecture fonctionnelle du RCG

Un exemple de distribution d'horloge à destination du sous-système d'interconnexion est décrit dans la Figure III.10. En fonctionnement normal, l'horloge *sys_clk* possède une fréquence double de celle du bloc CPU (500 MHz), la fréquence de l'horloge *hclk* des bus AXI est le tiers de l'horloge CPU et l'horloge *pclk* possède une fréquence moitié par rapport à hclk. En mode RESET, l'horloge *sys_clk* est issue du signal osci2 (32 kHz correspondant au mode *DOZE*) puis, lorsque le système démarre, l'horloge utilisée est *osci1* correspondant au mode *SLOW* (24 MHz). Ce dernier mode est aussi utilisé lors des transitions de fréquence sur osci2 pour éviter les *glitches*. La distribution d'horloge vers les autres sous-systèmes du SPEAr1340 est basée sur le même principe de sélection de signaux d'horloge source et de diviseurs permettant de définir les fréquences des horloges appliquées aux différents blocs fonctionnels.

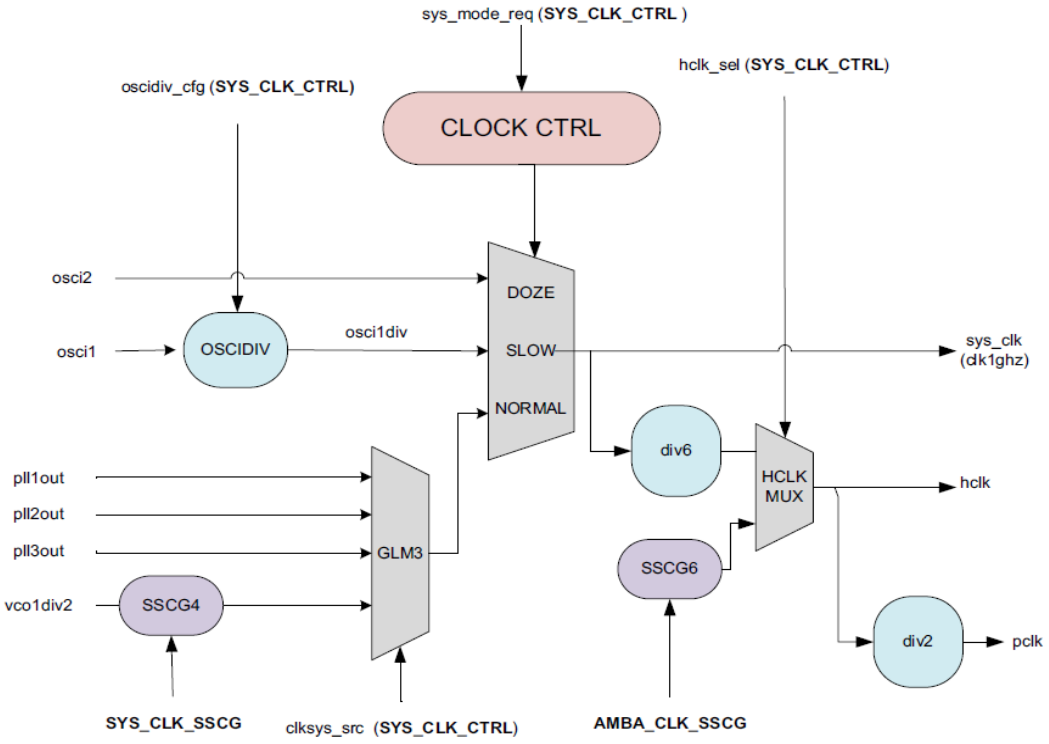


Figure III.10 – Distribution d'horloge à destination de l'interconnexion

Dans le circuit SPEAr1340, un sous-système peut avoir son propre CM. Par exemple, le CortexA9 possède son CM comme le montre la Figure III.11 dont l'horloge d'entrée CLK1GHZ est connectée à l'horloge SYS_CLK (1GHz ou 1,2 GHz au maximum). Le CM fournit aux CPU une horloge de fréquence moitié par rapport à SYS_CLK. Les périphériques internes à ce bloc CortexA9 (e.g. WD0) ont une horloge égale au quart de la fréquence SYS_CLK. Les horloges des ports AXI0, AXI1 et ACP ont une fréquence de 166 MHz et celle des ports APB est de 83MHz.

2.3 Conclusion de l'analyse des deux architectures OMAP4470 et SPEAr1340

Même si les deux architectures OMAP4470 et SPEAr1340, présentées dans les paragraphes précédents, possèdent toutes les deux des mécanismes de gestion de puissance, les principes de ces mécanismes sont assez différents. En effet, dans l'OMAP4470 il y a une partie importante du système dont la gestion des états de puissance est organisée en mode réactif alors que dans le SPEAr1340 la gestion centralisée contient un nombre réduits d'états de puissance qui sont

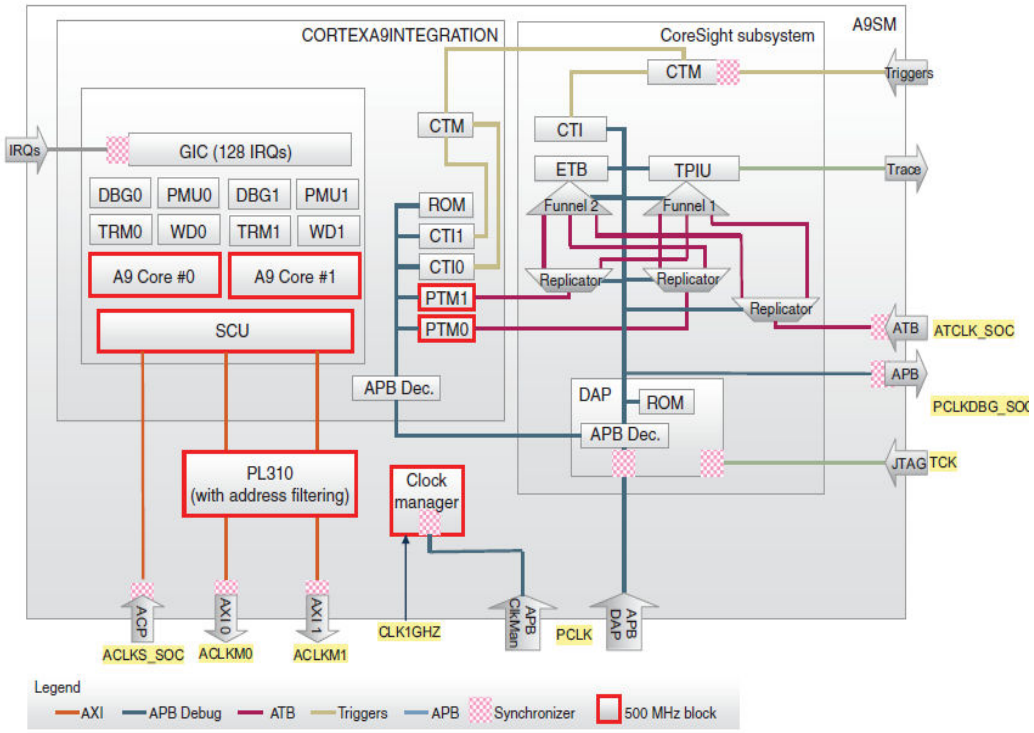


Figure III.11 – Le sous-système CortexA9 avec un *Clock Manager*

la conséquence du nombre réduit de scénarios fonctionnels d'utilisation prévus.

Par ailleurs, les deux architectures possèdent des similitudes concernant leur architecture de gestion de puissance. Ainsi, la structuration d'un arbre d'horloge est composé de fonctionnalités communes : sources ou générateurs (DPLL), diviseurs (pour produire les fréquences à appliquer aux blocs), multiplexeurs (pour sélectionner les bonnes fréquences en fonction des états fonctionnels), *clock gating* de manière à pouvoir supprimer la puissance dynamique des blocs fonctionnels non actifs. Les blocs dans un même *Clock Domain* peuvent avoir des horloges avec des fréquences différentes mais ces horloges sont synchrones et sont issues d'une même horloge source sur laquelle sont appliqués des facteurs de division différents. Ces fonctionnalités sont regroupées dans un CM.

Nous nous intéressons dans la suite à illustrer comment un arbre d'horloge peut être structuré, modélisé et contrôlé dans une approche de description à un niveau transactionnel.

3 Modélisation des *Clock Domains* au niveau SystemC-TLM

Avant d'entamer la définition des concepts utilisés dans la modélisation d'un arbre d'horloge au niveau transactionnel, nous présentons brièvement l'intérêt d'expliciter en simulation une décomposition d'une architecture matérielle en *Clock Domain* (CD).

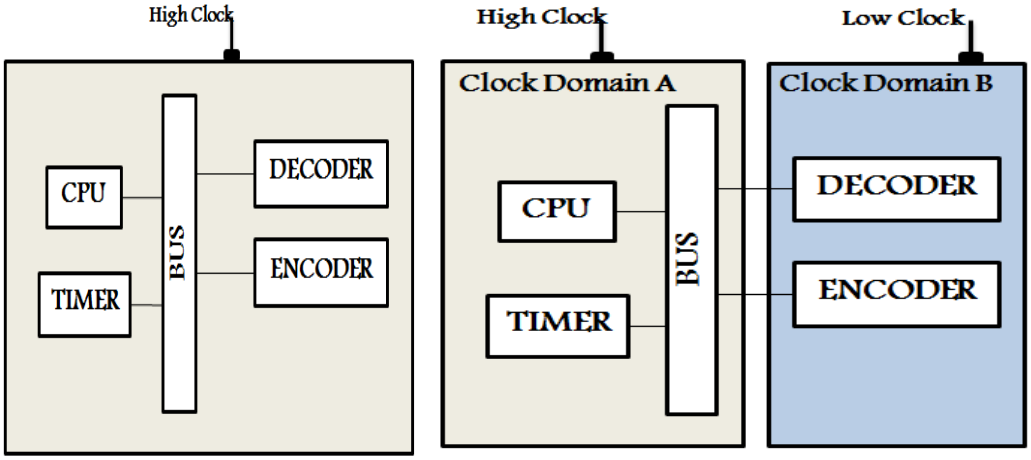


Figure III.12 – Exemple de partitionnement en *Clock Domains*

Prenons l'exemple de l'architecture simple illustrée sur la Figure III.12(a) dont le but est d'encoder ou de décoder un flux d'échantillons. Cette architecture est composée de cinq blocs IP alimentés par la même source d'horloge unique. À l'exécution, tous les blocs sont actifs et cadencés à la même fréquence. Ainsi, la consommation dynamique du système ne varie que sur la base de l'activité interne d'un bloc IP. Cependant, selon les exigences de latence et les exigences fonctionnelles de l'application, tous les blocs IP ne doivent pas nécessairement être cadencés à la même fréquence de fonctionnement. Par conséquent, l'architecture peut être divisée par exemple en deux CD (Figure III.12(b)) où l'encodeur et le décodeur sont alimentés par une source d'horloge commune ayant une fréquence plus basse que celle du CPU et du bus. Outre la réduction de la puissance dynamique dissipée obtenue, cette décomposition explicite en *Clock Domain* permet dans une simulation de décrire une gestion structurée de l'arbre d'horloge. En effet, au moment d'un changement d'OPP (*Operating Performance Point*), seules les fréquences de fonctionnement des blocs IP du ou des *Clock Domains* concernés subissent une modification (et donc les performances temporelles associées), les fréquences des autres blocs IP ne devant

pas être affectées. Une mise à jour de l'estimation de la consommation de puissance dynamique peut être aussi effectuée par *Clock Domain* dès qu'un changement de fréquence intervient, ce qui aide à l'évaluation et à la vérification du système.

Toutefois, cette notion de décomposition en CD est actuellement peu formalisée en particulier par l'absence de standard sur ce point. De nombreux travaux s'intéressent au problème de la synchronisation entre CD (*Clock Domain Crossing*) et se focalisent sur l'interface entre les unités communicantes alimentées par des horloges différentes. Au niveau d'abstraction considéré dans cette thèse les horloges ne sont modélisées que par leur fréquence (ou les variations de fréquence). Le problème de synchronisation d'horloges entre *Clock Domains* n'a pas de sens et est donc ignoré ici.

Néanmoins, la littérature ne propose que rarement une définition du concept de CD. Dans [LK09], la notion du CD proposée par *Mentor Graphics* est la suivante : « Un CD est composé de toute la logique séquentielle d'un système contrôlé par une horloge unique, ou par cette horloge inversée ou par une horloge produite par division de cette horloge ».

Freescale [LHDX10] définit un CD comme « étant un sous-ensemble du système soit contrôlé par une horloge unique ou des horloges qui ont des relations de phase constantes. Un tel sous-ensemble est dit synchrone. De plus, des sous-ensembles qui ont deux ou plusieurs horloges avec des fréquences ou des relations de phase variables sont considérés comme de type asynchrone et constituent différents domaines d'horloge.» Cette définition est plus générale que la précédente étant donné que les horloges peuvent être déphasées, à condition que la phase reste constante. Dans les approches qui visent une conception orientée GALS comme dans [TM05], chaque CD possède son propre générateur d'horloge local ce qui définit ainsi un bloc synchrone au sein d'une architecture GALS.

Toutes ces définitions de CD n'impliquent aucune relation avec la notion de PD contrairement à un CD dans l'architecture OMAP4470 qui est nécessairement inclus dans un PD. Suite à nos analyses, nous avons constaté que la relation d'inclusion présente dans le circuit OMPA4470 est une bonne approche pour faciliter la gestion et la conception du système mais que cette relation ne peut pas être considérée comme une généralité.

Par conséquent, dans notre approche de modélisation proposée nous ne faisons pas apparaître de relation d'inclusion telle qu'utilisée dans le circuit OMAP4470. Nous définissons un *Clock Domain* comme un ensemble de modules fonctionnels qui possèdent à tout instant une seule source d'horloge sur laquelle des opérations de division peuvent être effectuées mais qui conduisent à des signaux toujours synchrones ou à une phase constante au niveau des modules du CD. Entre deux CD, aucune hypothèse n'est faite sur les horloges associées à ces CD.

3.1 L'environnement *ClkARCH*

Dans ce paragraphe, nous décrivons notre approche qui offre un moyen de superposer à un modèle d'architecture fonctionnelle (par exemple un prototype virtuel) décrite en SystemC-TLM, un modèle exécutable représentant un système de distribution d'horloges avec son contrôle associé (*clock intent*) et muni de capacités de vérification.

3.1.1 Spécification du *clock intent* au niveau transactionnel

Comme cela a été illustré avec les deux architectures OMAP4470 et SPEAr1340, la distribution d'horloge est structurée principalement avec des générateurs d'horloge ou des sources d'horloge (les DPLL ou des signaux externes) et des unités de contrôle d'horloge (*Clock Manager*, CM) qui assurent trois fonctions. Premièrement, une fonction qui permet de définir pour chacune des horloges générées les fréquences d'horloges à appliquer aux modules fonctionnels. Ces fréquences sont définies en appliquant des facteurs de division a priori prédéterminés. Deuxièmement, il s'agit de sélectionner la fréquence d'horloge à appliquer aux modules fonctionnels depuis l'ensemble des horloges disponibles (signaux externes, horloges générées, horloges produites par les diviseurs). Enfin, la troisième fonction effectue du *clock gating* sur chacune des horloges ainsi produites.

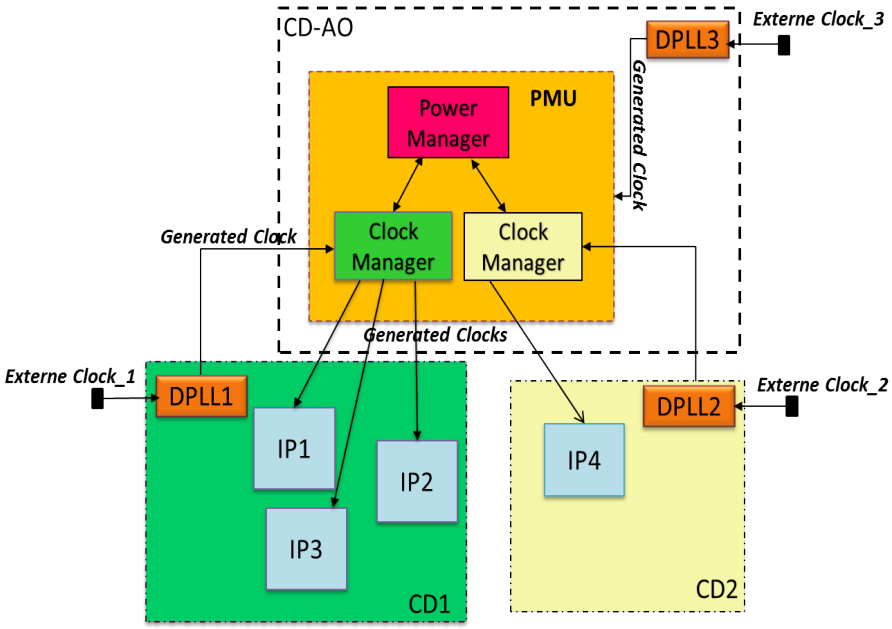


Figure III.13 – Architecture fonctionnelle augmenté par *clock intent*

Cette structuration apparaît assez générale et nous proposons de la considérer pour définir une modélisation d'arbre d'horloge dans une architecture matérielle décrite en SystemC-TLM. La modélisation est présentée par un exemple dans la Figure III.13. Cette modélisation de distribution d'arbre d'horloges ainsi que son contrôle associé sont développés dans l'environnement *ClkARCH*. L'environnement *ClkARCH* consiste en une librairie logicielle statique composée d'un ensemble de classes permettant de faciliter l'instrumentation d'un prototype virtuel décrit en SystemC-TLM avec les caractéristiques d'un *clock intent*. Cette librairie est générique et réutilisable et son utilisation ne dépend pas du prototype virtuel cible à instrumenter. Toutefois, le prototype virtuel doit être décrit suivant le standard SystemC-TLM et structuré comme un ensemble d'instances de blocs IP.

La Figure III.14 montre la structure et les relations entre les différentes classes qui composent *ClkARCH*. Dans *ClkARCH*, trois sous-ensembles peuvent être distingués dont le premier *"High-level clock Intent Specification"* est un groupe de classes C++ qui permet la spécification et la construction d'un arbre d'horloge. Le deuxième *"Clock control objetcs"* regroupe deux modules SystemC-TLM dédiés pour le contrôle et la gestion de la distribution d'horloges dans le prototype virtuel. Le troisième *"Clock Management Verification + Clock Monitoring"* est un groupe de classes C++ dédié pour la vérification de la structure et la gestion de l'arbre d'horloge. Ce dernier groupe permet également l'estimation de la consommation d'énergie durant la simulation. Dans la suite, nous expliquons les principales caractéristiques de *ClkARCH*.

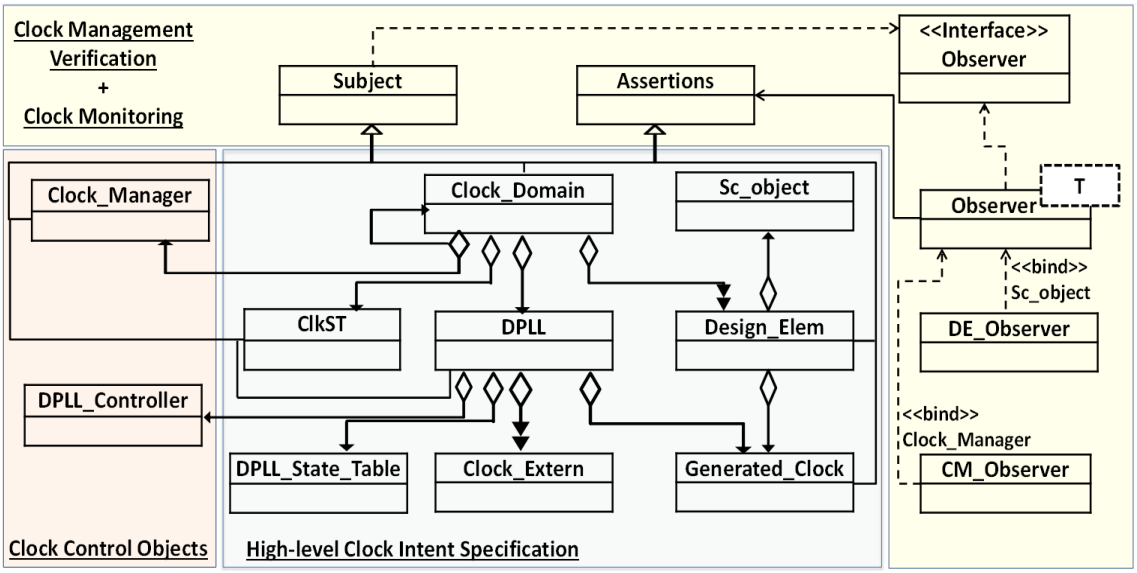


Figure III.14 – Diagramme de classes de *ClkARCH*

Notre environnement favorise la séparation des préoccupations entre les modèles fonctionnels et non fonctionnels. Ainsi, pour relier les concepts de distribution d'horloge avec l'architecture fonctionnelle SystemC-TLM, nous définissons un objet C++ *Design_Element* (DE) [MPA11] qui fait référence à un module SystemC représentant un bloc IP de l'architecture fonctionnelle comme l'illustre l'exemple sur la Figure III.19, où le bloc IP1 est lié au *Design_Element* DE1. Un DE est associé à un *Clock Domain* (Figure III.14). Par conséquent, il doit être attaché simultanément à un objet SystemC (*sc_module*) du modèle fonctionnel et à un *Clock Domain* instancié à partir de *ClkARCH*. Il joue le rôle de passerelle entre la conception du *clock intent* construite en utilisant *ClkARCH* et la conception fonctionnelle SystemC-TLM.

3.1.1.1 Notion du *Clock Domain* : Un *Clock Domain* dans *ClkARCH* est un objet C++ qui contient un ou plusieurs DE possédant à tout instant qu'une seule source d'horloge sur laquelle des opérations de division peuvent être effectuées mais qui conduisent à des signaux toujours synchrones ou à une phase constante au niveau des DE du CD. Au niveau TLM il n'y a pas de vérification effectuée sur les différentes phases des horloges, nous supposons donc qu'au sein d'un CD cette phase est bien constante. Ainsi, définir un CD par les DE qui y sont inclus permet de contrôler ces derniers et d'appliquer le *clock gating* indépendamment des autres DE appartenant à d'autres *Clock Domains*. Contrairement à la structure hiérarchique des *Power Domains* définie par UPF [upf], nous n'introduisons pas de hiérarchie entre les *Clock Domains*. En effet, si un CD-A fournit une horloge vers un CD-B, l'horloge en entrée du CD-A sera synchrone (ou avec une phase constante) avec toutes les horloges produites par le CD-B. Toutes ces horloges étant synchrones ou à une phase constante, l'ensemble ne constitue alors qu'un seul CD suivant la définition donnée ci-dessus. Du point de vue de l'implémentation du modèle, nous avons choisi de définir un CD comme étant de type *container*. Ce CD de type *container* n'induit pas une composition hiérarchique entre les CD mais il est défini pour mieux structurer notre méthodologie.

3.1.1.2 Modèle de DPLL : Nous associons pour un *Clock Domain* une seule DPLL qui représente sa source primaire d'horloge et un seul *Clock Manager* qui permet de contrôler la distribution d'horloge à ses *Design_Elements*. Dans l'OMAP4470, il n'y a que deux CM (CM1 et CM2) alors que le nombre de CD est largement plus important. Ce choix de deux CM est lié au fait que CM1 est dans un *Power Domain* de type *Always-On* alors que CM2 est dans un *Power Domain* qui peut avoir son alimentation électrique coupée. Cependant, la distribution d'horloge effectuée dans l'OMAP4470 fait bien apparaître une séparation fonctionnelle entre les entités de multiplexage et de division qui opèrent sur des signaux d'horloge synchrones.

Aussi, nous pouvons considérer que les CM1 et CM2 de l'OMAP4470 sont des regroupements structurels des CM fonctionnels par CD et donc nous avons choisi d'associer un seul CM à une DPLL et donc à un *Clock Domain*. Toutefois, il apparaît dans l'OMAP4470 que des signaux issus de deux DPLL peuvent être utilisés après multiplexage comme horloge d'un bloc. Néanmoins, ce cas arrive lorsque ces deux DPLL possèdent une même horloge de référence externe et donc les horloges générées par les DPLL sont synchrones une fois qu'elles sont verrouillées. Dans un but de structuration et pour éviter des situations erronées où un CM utiliserait en entrée des horloges asynchrones nous n'avons pas souhaité prendre en compte ce cas. Il peut bien sûr être représenté directement en SystemC-TLM en insérant un bloc ad-hoc qui réalise cette fonction de multiplexage entre les horloges issues de deux CM.

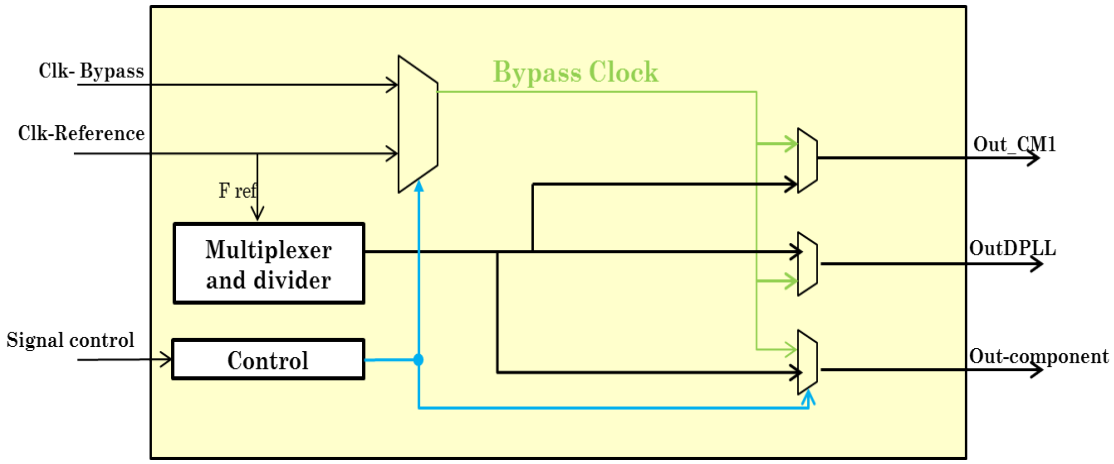


Figure III.15 – Structure d'une unité de DPLL générique dans *ClkARCH*

L'abstraction d'une DPLL est définie comme une classe (Figure III.14) dans notre approche de modélisation d'arbre d'horloge au niveau transactionnel. Sa structure est montrée dans la Figure III.15. Une DPLL reçoit au plus deux entrées d'horloges externes. La première est une horloge de référence (Clk-Reference) utilisée pour fournir une horloge de sortie de type *Generated_Clock* (GC) sur la base des facteurs de division et de multiplication définis pour cette DPLL. Le second signal d'horloge d'entrée est une horloge de dérivation (Clk-Bypass) utilisée quand une DPLL est en mode de verrouillage ou de réinitialisation. Dans ces modes, les blocs appartenant à un même *Clock Domain* fonctionnent à la même fréquence d'horloge. Le temps de verrouillage est défini comme une pénalité de temps dans la DPLL. Pendant ce temps, tous les blocs du CD concerné reçoivent la fréquence d'horloge de dérivation. Dans l'autre cas où une DPLL ne dispose que d'une horloge d'entrée unique, cette horloge est considérée à la

fois comme une horloge de référence et de dérivation. À partir de la fréquence de l'horloge d'entrée de la DPLL, les fréquences des horloges de sortie sont déterminées par des opérations en arithmétique entière. L'horloge produite en sortie de la DPLL suit l'équation III.2.

$$Out_{Clk} = F_{ref} * \frac{M}{N} \tag{III.2}$$

Où M et N sont des nombres entiers qui représentent respectivement un facteur de multiplication et un facteur de division. La DPLL peut produire une horloge pour trois différentes destinations : à un CM afin qu'il distribue les horloges à l'intérieur du CD, à un bloc directement dans le cas où le bloc est alimenté avec une fréquence stable (par exemple dans un PD de type *Always-On*) et enfin à une autre DPLL de l'architecture (ce cas est présent dans l'OMAP4470). Une DPLL a la capacité d'effectuer du *clock gating* sur les horloges produites. Un contrôle peut alors être réalisé sur tout l'arbre d'horloge issu d'une DPLL. Pour ce faire, nous avons modélisé un *DPLL_Controller* (c.f. Figure III.13).

3.1.1.3 Le *DPLL_Controller* : Un *DPLL_Controller* envoie d'une part des contrôles pour sélectionner les horloges de dérivation ou de référence et d'autre part, les demandes de changement des valeurs M et N pour chaque DPLL concernée lorsqu'il reçoit une requête de changement d'OPP (par exemple, pour appliquer la technique de DVFS décrite dans la section 2.5.3.3). Dans l'OMAP4470, le PRM qui est le composant du PRCM assurant le contrôle des DPLL, joue le rôle du *DPLL_Controller*. Certaines fonctions du PRM sont ignorées ici comme le recalibrage des DPLL qui doit intervenir lorsque la température varie de manière importante.

Par ailleurs, un *DPLL_Controller* gère les transitions d'un mode de puissance à un autre. Ces modes sont définis dans une *DPLL State Table* (DST). Cette table contient ainsi les OPP qui peuvent être sollicités par le logiciel applicatif ou le logiciel système, par exemple par l'intermédiaire de la fonction *CPUFreq*() de Linux. La Figure III.16 montre un exemple de cette

	DPLL1		DPLL2		DPLL 3	
	M	***N***	***M***	***N***	***M***	***N***
G_State_1:	1	2	4	5	4	1
G_State_2:	0	0	1	0	2	4

Figure III.16 – Exemple d'une *DPLL State Table*

table statique et bidimensionnelle. Les colonnes d'une DST représentent des états d'horloge des DPLL en définissant les valeurs respectives des facteurs de division et de multiplication M et N permettant de générer pour chaque DPLL l'horloge de sortie à la fréquence désirée à partir de son horloge d'entrée de référence. Ainsi, les lignes de la DST représentent des modes fonctionnels et également des modes de puissance globale d'un système. Ces modes fonctionnels sont sélectionnés soit au niveau application soit au niveau du système d'exploitation.

Figure III.17 – Représentation du *Clock Manager*

3.1.1.4 Le *Clock Manager* : Dans *ClkARCH*, nous associons un seul CM par CD dont sa structure est illustrée sur la Figure III.17. Le CM est alimenté par une horloge (*Generated Clock input*) fournie par la DPLL du même CD. Il génère des horloges de type GC (Generated Clock Out_1, ..., Generated Clock Out_n) aux blocs fonctionnels contenus dans le CD associé, une horloge par DE.

Les CM fournissent la valeur de la fréquence d'horloge aux blocs fonctionnels de l'architecture. La valeur de la fréquence fournie sur une sortie d'un CM correspond à la fréquence de l'horloge d'entrée du CM sur laquelle peut être appliquée un facteur de division représenté par un entier positif et en puissance de deux. La fréquence de l'horloge d'un bloc IP peut donc prendre des valeurs discrètes définies à partir des paramètres de la DPLL source et des facteurs de division disponibles dans le CM. Elle peut être nulle en appliquant la fonction du *clock gating* au niveau du CM ou de la DPLL source. Cette fréquence est ajustée afin d'offrir un bon compromis entre la puissance dynamique consommée et la performance du système. Toutefois, pour que l'évaluation des performances soit cohérente, les délais renseignés dans les modules fonctionnels pour représenter les temps de traitement doivent être paramétrés par la fréquence d'horloge à laquelle ces traitements sont exécutés. Ceci est impératif pour que les comportements en temps du module soient corrects par rapport à la stratégie de *Power Management*. Nous décrivons ces délais dans les modèles fonctionnels SystemC-TLM sous forme d'un nombre de cycles de manière à être indépendant dans le modèle de la fréquence d'horloge.

Ceci constitue une contrainte sur les modèles fonctionnels des blocs IP, mais il s'agit en réalité d'une règle logique de modélisation à observer.

3.1.1.5 Le *Power Mangement Unit* (PMU) : Les CM sont situés dans le *Power Mangement Unit* (PMU) et connectés au *Power Manager* (PM) comme illustré sur la Figure III.13. Le CM reçoit du PM un signal de contrôle qui demande un ajustement de l'état du CD : appliquer la fonction du *clock gating* ou modifier les fréquences d'horloge avec de nouveaux facteurs de division. Après le traitement de la demande, le CM renvoie un acquittement au PM à travers le signal *Clock Activity Status*. Le CM peut signaler au PM également à travers le même signal l'état du CD. Par exemple, celui-ci peut être dans un état actif sur au moins une horloge de sortie ou en mode *clock gated* sur toutes les horloges générées. Avec cette information, il est par exemple possible de décider si le *Power Domain* qui contient des modules de ce CD peut être mis globalement en état *power gated*.

	Clock Domain 1			**Clock Domain 2**			**Clock Domain 3**	
	DE1	***DE2***	***DE3***	***DE4***	***DE5***	***DE6***	***DE7***	***DE8***
C_State_1:	1	2	1	4	0	1	0	1
C_State_2:	0	1	2	0	0	0	2	4
C_State_3:	1	0	2	4	2	2	0	0

Figure III.18 – Exemple de la *Clock State Table*

Afin de définir les différents modes de puissance que peuvent prendre les *Clock Domains* d'une architecture, nous introduisons une *Clock State Table* (ClkST) au sein du PMU. Cette table est similaire à la *Power State Table* définie dans UPF [upf]. Elle participe à la définition d'une stratégie de gestion de la puissance consommée du système basée sur l'adaptation relative des horloges des *Clock Domains* tout en permettant d'obtenir les performances souhaitées. En effet, elle identifie pour chaque bloc ses valeurs de fréquences possibles durant l'exécution d'un scénario. Comme le montre la Figure III.18, une ClkST est une table statique bidimensionnelle qui contient les paramètres nécessaires pour définir l'état de chaque CD. Les colonnes d'une ClkST représentent des modes de puissance locaux des DE. Un mode de puissance local définit l'état d'horloge de chaque bloc (*Design_ Element*) en spécifiant son facteur de division. Ce facteur de division est appliqué par le CM sur l'horloge source du CD afin de générer l'horloge pour le *Design_ Element* correspondant pour un mode de puissance déterminé. Lorsque le

facteur est nul, la fonction de *clock gating* est appliquée. Les lignes d'une ClkST représentent les modes de puissance globaux du système. En effet, il y a une correspondance directe entre l'état fonctionnel souhaité et une ligne de cette table.

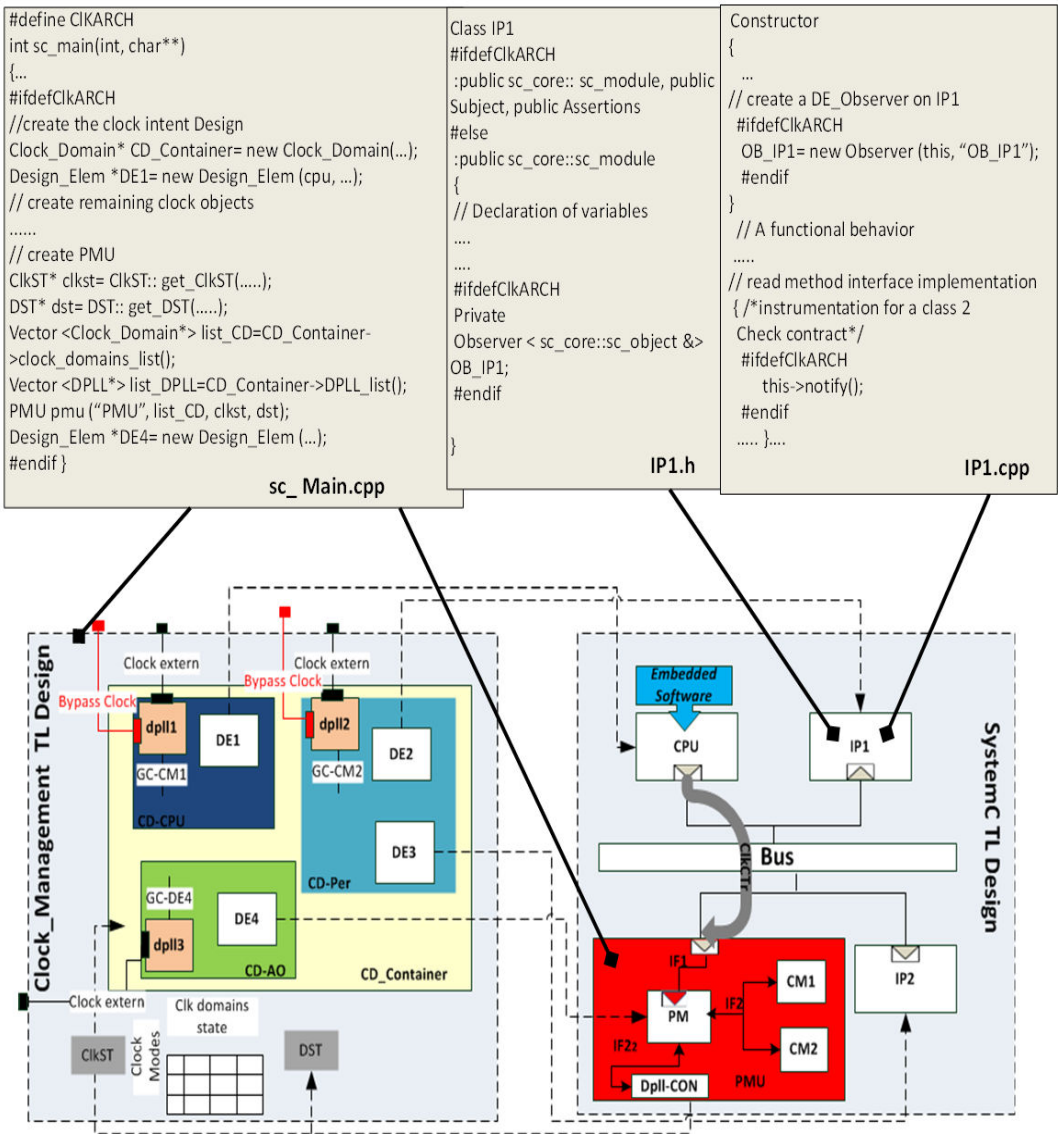


Figure III.19 – Approche basée sur l'instrumentation

3.1.1.6 Vue générale d'un modèle de *clock intent* : Nous avons présenté les concepts permettant la structuration et la gestion d'un arbre d'horloge au niveau transactionnel. Les interactions entre les composants d'un *clock intent* construit en utilisant *ClkARCH* et un système fonctionnel sont illustrées sur la Figure III.19. Une conception du *clock intent* est obtenue en

augmentant la classe principale (sc_main) du système fonctionnel avec un code supplémentaire (*Power_Main*) qui utilise la bibliothèque *ClkARCH*. Cette partie de code ajoutée est précédée par une déclaration # *ifdef ClkARCH* comme le montre la Figure III.19. Comme nous pouvons le voir dans cette figure, les composants du *clock intent* sont instanciés et des valeurs sont affectées à certains de leurs attributs au moment de l'instanciation. Nous notons que cette instanciation est réalisée dans un ordre spécifique pour répondre aux règles de composition de dépendances entre les différents objets du *clock intent* (Figure III.14). Notons enfin que le PMU est le seul composant SystemC-TLM ajouté dans le modèle fonctionnel ce qui limite l'insertion de code intrusif dans le code fonctionnel.

3.1.2 Estimation et Analyse de puissance

Pour effectuer l'estimation de la consommation de puissance lors d'une simulation fonctionnelle, un objet C ++ *Power_Monitor* est automatiquement instancié dans la section principale SystemC-TLM du modèle au moment de l'instanciation du CD de type *container*. Lorsqu'un événement de changement du mode de puissance se produit (changement de la valeur de la fréquence), cet objet *Power_Monitor* est alerté lors de la simulation pour mettre à jour la consommation de puissance relative au CD concerné.

Pour capturer ces événements de changement du mode de puissance, *ClkARCH* implémente une solution évolutive et un mécanisme facile à utiliser basé sur le parton de conception observateur C++ (C++ *observer design pattern*). Comme le montre la Figure III.14, nous incluons également une classe de CM_Observer et une classe de DE_Observer. Ces classes sont dérivées de la classe générique et du *template* respectivement sur les classes *Clock_Manager* et *Design_Elem*.

En conséquence, la consommation de puissance est mise à jour d'abord au niveau DE en appliquant les équations II.1 et II.2. Au niveau *Clock Domain*, chaque CD a une consommation de puissance totale P_{CD_total} qui représente la somme de la puissance dynamique $P_{CD\_dynamic}$ et la puissance statique P_{CD_static}, elles-mêmes calculées suivant l'équation III.3 où NbDE représente le nombre de DE contenus dans le CD concerné.

$$\forall j / 0 \leq j \leq NbCD$$

$$P^{j}_{CD\_total}(t) = \sum_{i=0}^{NbDE(j)\neq i} P^{i}_{DE\_static}(t) + \sum_{i=0}^{NbDE(j)\neq i} P^{i}_{DE\_dynamic}(t) \qquad \text{(III.3)}$$

Les données liées à la puissance et à la technologie comme l'activité de commutation et la capacitance sont attachées aux objets DE et utilisés par les *observers* pour mettre à jour les équations de consommation de puissance $P_{DE\_dynamic}$ et P_{DE_static}.

3.2 La méthodologie basée sur les *Clock Domains*

Dans cette section, nous proposons une méthodologie basée sur la modélisation des *Clock Domains* afin d'insérer une stratégie de gestion de la distribution d'horloge dans un modèle fonctionnel pur décrit en SystemC-TLM qui ne contient au départ aucune gestion de puissance. La Figure III.20 représente la démarche globale de notre méthodologie. Cette méthodologie se compose de quatre étapes dont trois sont exécutées successivement et une est traitée en parallèle avec les étapes précédentes. Ces différentes étapes sont complémentaires et dépendent toutes de *ClkARCH*. Les principales caractéristiques et les objectifs de chaque étape sont décrits dans la suite.

3.2.1 Étape de la spécification d'un *clock intent*

Avant de commencer cette étape de spécification, le concepteur doit avoir une idée claire sur le comportement fonctionnel de l'application exécutée sur le prototype virtuel SystemC-TLM et suivant les différents scénarios possibles de cette application. Pour cela, le concepteur analyse tout d'abord les échanges de données entre les composants fonctionnels au cours de simulations effectuées avec ces différents scénarios. Cette analyse permet de comprendre quand et combien de fois chaque composant est activé selon le scénario d'application exécuté. Ces informations avec la connaissance des nombres de cycles d'exécution des traitements effectués par ces composants aident à déterminer les fréquences et les rapports de fréquences entre les composants de manière à satisfaire des objectifs de performance.

Capturer les traces des transactions facilite également la définition des dépendances entre les activités des composants mettant en évidence les corrélations possibles entre les blocs matériels qui devront être prises en compte par la stratégie de gestion de puissance. Ayant cet ensemble d'information, le concepteur formule une proposition sur les différentes fréquences de fonctionnement à affecter aux différents blocs. Par conséquent, il peut spécifier une première architecture d'horloge pour le prototype virtuel SystemC-TLM supportant la distribution d'horloges définies. Cela se fait par l'instanciation des objets d'horloges appropriés à partir de la librairie *ClkARCH*. Ces objets sont superposés avec les blocs IP de l'architecture matérielle via les objets *Design_Element* (DE).

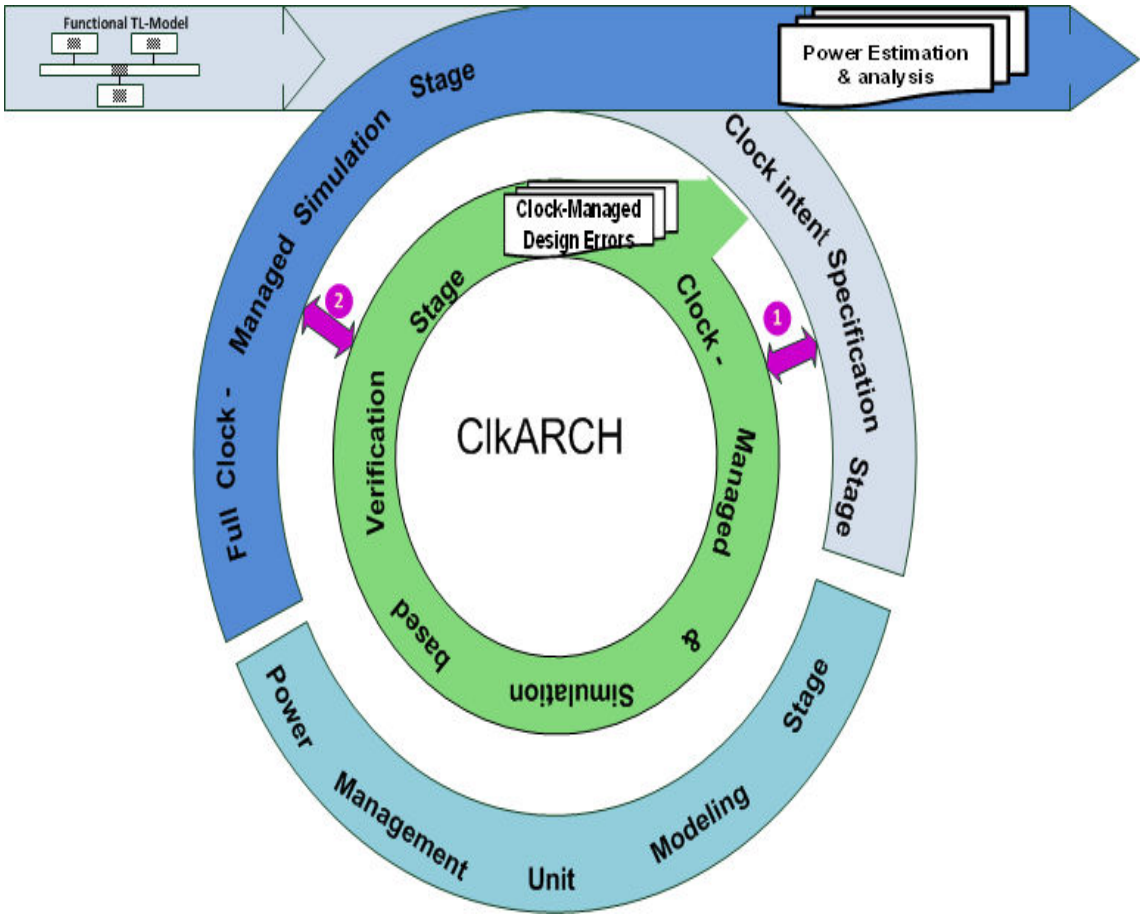
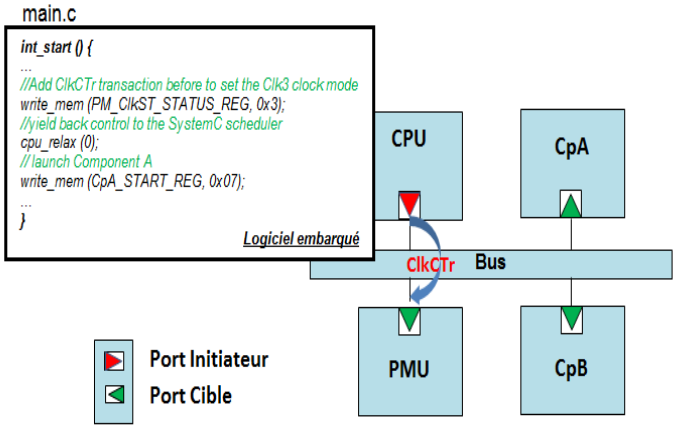


Figure III.20 – Flot global de la méthodologie basée sur des *Clock Domains*

À cette étape, le concepteur partitionne le prototype virtuel en CD et met en place les sources d'horloge ainsi que les horloges générées (GC). Selon l'analyse des traces des activités et la spécification de la distribution d'horloge correspondante, une stratégie de gestion de puissance du système doit être définie en spécifiant les tables *Clock State Table* (ClkST) et la *DPLL State Table* (DST). En effet, le flux logiciel issu de l'exécution de l'application suivant ses différents scénarios sur le prototype virtuel SystemC-TLM représente des informations pertinentes afin de préciser les états globaux d'horloges renseignés dans les tables ClkST et DST (les OPP). Enfin, ces informations aident le concepteur à associer les OPP à utiliser pour les CD en fonction des tâches à exécuter durant l'exécution de chaque scénario de l'application.

3.2.2 Étape de la modélisation de l'unité de gestion de puissance

Traditionnellement, la gestion de puissance d'un système est basée essentiellement sur une unité de gestion centralisée au sein d'un système. Contrairement à l'approche classiquement considérée [Ta11] dans laquelle cette unité intègre l'essentiel de l'intelligence nécessaire à la gestion de la puissance dissipée, cette unité de gestion de puissance (*Power Management Unit*, PMU) joue ici le rôle d'une interface entre le matériel induit par la structuration en *Clock Domains* et le logiciel embarqué.



(a) Exemple d'une transaction de contrôle (ClkCTr) relative à la ClkST

	CD_1		CD_2	
	Bus	CPU	CpA	CpB
Clk1	2	1	2	0
Clk2	1	1	0	1
Clk3	1	2	2	4

(b) Exemple de la *Clock State Table*(ClkST)

Figure III.21 – Exemple d'un scénario appliquant la stratégie basée sur les *Clock Domains*

Le PMU est modélisé comme un bloc fonctionnel SystemC-TLM qui communique à travers le bus par des transactions TLM avec les autres blocs IP de la plateforme comme le montre la Figure III.21(a). Cette unité sert de relai à la mise en œuvre de la stratégie de gestion de puissance basée sur l'exploitation des états des *Clock Domains*. Cette stratégie permet de contrôler le mode de puissance d'un CD adapté au scénario applicatif en ajustant les modes de puissance de ses composants. Basculer d'un mode de puissance à un autre correspond alors à une demande de gestion de puissance représentée comme une transaction TLM issue d'un initiateur

dans le prototype virtuel (par exemple, le CPU sur lequel s'exécute le logiciel embarqué). Ce type de demande est notée *ClkCTr* (voir Figure III.21).

Par ailleurs, cette étape consiste également à intégrer le PMU dans le modèle fonctionnel SystemC-TLM. Généralement, un PMU appartient à un CD toujours actif *CD-AO* afin d'être réceptif et de réagir à toutes les transactions entrantes de type *ClkCTr*. La structure d'un PMU, comme illustrée sur la Figure III.22, est composée d'un gestionnaire d'alimentation (PM), d'un *DPLL_Controller* et d'un ensemble de CM qui sont modélisés comme des sous-blocs SystemC-TLM. Dans cette structure, c'est le PM qui contient les différentes tables représentant les états possibles des *Clock Domains* afin de commander les *Clock Managers* appropriés pour modifier les modes de puissance de leurs CD (la partie *PM Command Dispatcher* dans la Figure III.22). Un CM doit être associé à chaque CD pour changer son mode de puissance sous la demande du PM. Le PM envoie des demandes au *DPLL_Controller* pour que ce dernier ajuste la fréquence de sortie des DPLL concernées. Tous ces comportements sont directement définis dans les modèles de ces composants de *ClkARCH*.

La communication entre un CM et un CD d'une part et entre des DPLL et un *DPLL_Controller* d'autre part doit être précisée avec l'objectif de ne pas impacter sensiblement la vitesse de simulation du système complet. En fait, pour compléter la modélisation d'un PMU, un ensemble d'interfaces de communication est proposé. Comme présentées dans la Figure III.22, cinq interfaces différentes permettant de supporter les échanges de données et de contrôle sont modélisées.

- Une interface fonctionnelle nommée IF1 est mise en place entre le PMU et les autres modules fonctionnels de la plateforme. Sur cette interface, les transactions fonctionnelles sont principalement transmises pour initialiser le PMU ou configurer ses registres (CSR, *Control and Status Registers* de la Figure III.22) afin de transmettre les transactions *ClkCTr* au PMU. Ces transactions peuvent également être utilisées pour déclencher une lecture d'un registre du PMU afin d'obtenir des informations sur l'état des CD.
- Deux interfaces internes utilisant le protocole requête/acquittement sont également définies :

  1. IF2 se situe entre le PM et un CM. Cette interface permet au PM de contrôler les activités du CM. En utilisant des signaux SystemC, le PM demande au CM de changer le mode de puissance du CD en ajustant les facteurs de division de ses DE. Puis, il reste en attente d'un acquittement du CM pour que celui-ci indique la fin de la modification du mode de puissance du CD.

2. L'interface IF22 se situe entre le PM et le *DPLL_Controller*. Elle permet au PM de gérer les activités du *DPLL_Controller*. Avec des signaux SystemC, le PM demande au *DPLL_Controller* de changer les facteurs M et N des DPLL. Ensuite, il attend un acquittement du *DPLL_Controller* de fin changement d'état. Dans notre cas, le DPLL_Controller utilise un temps constant pour réaliser ce changement de fréquence, un modèle plus précis peut y être développé par exemple connaissant les caractéristiques de la DPLL et celles des modules inclus dans le CD.

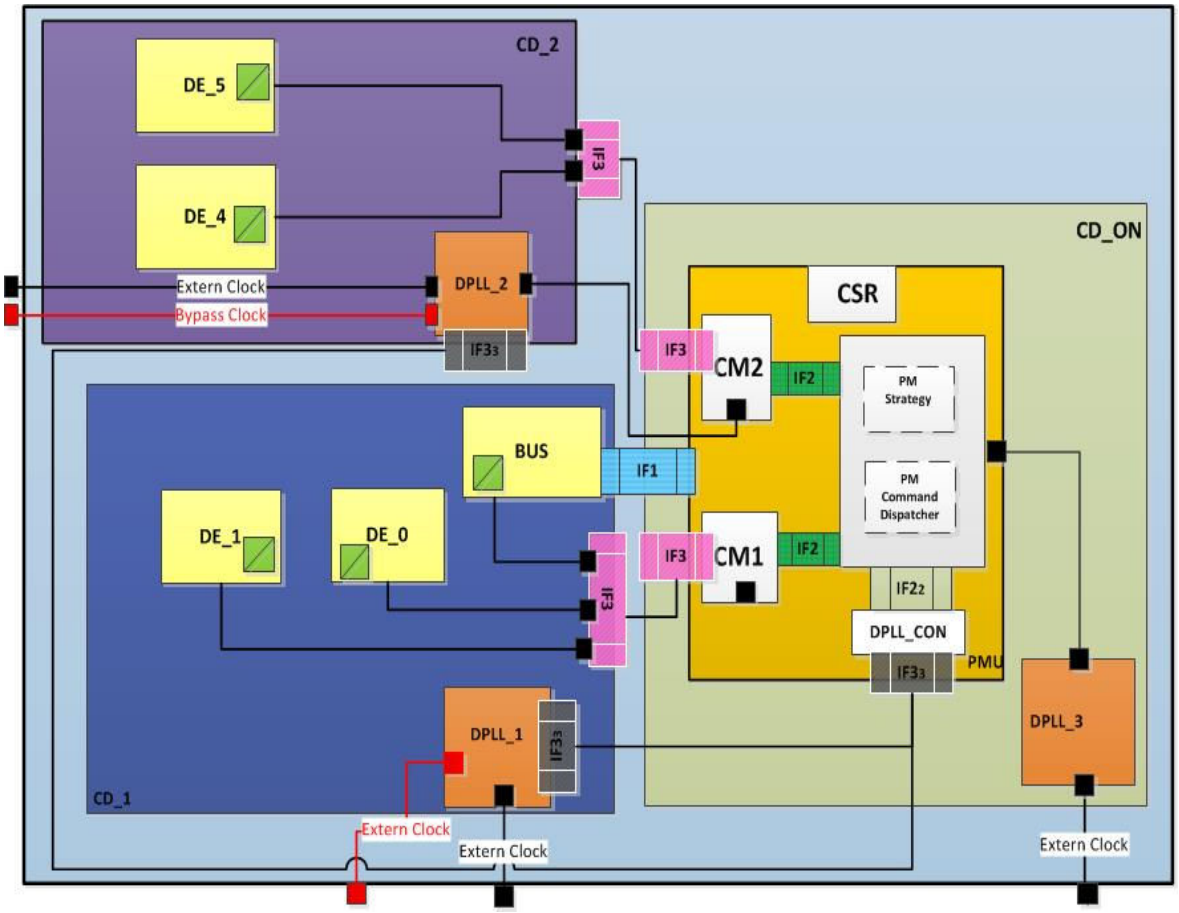


Figure III.22 – Architecture augmentée par les concepts du *clock intent*

– Deux interfaces de gestion d'horloge sont mises en place aussi dans la modélisation du PMU afin d'agir sur les DE des CD :

1. IF3 se situe entre le PMU et un CD afin de permettre le changement d'état des horloges à destination des DE. Comme le montre la Figure III.22, une communication avec ce type d'interface est assurée entre le CM et le CD.

2. IF33 se situe entre le PMU et la DPLL afin de sélectionner parmi les valeurs des horloges d'entrée (référence et *bypass*) celle qui est utilisée par la DPLL. Une communication peut être établie par ce type d'interface entre le *DPLL_Controller* (DPLLCON) et les DPLL comme l'illustre la Figure III.22.

Une stratégie de gestion de puissance s'appuyant sur les tables ClkST et DST est basée sur l'analyse des scénarios applicatifs. Ainsi, les tables ClkST et DST représentent des paramètres d'entrée au module PM et sont utilisées pour construire l'ensemble des modes de puissance utiles à la gestion de puissance. Ces modes utilisent directement les valeurs définies à la fois dans la ClkST et la DST. Chaque transition entre deux modes de puissance correspond à une configuration spécifique du registre CSR du PMU effectuée à travers la transaction *ClkCTr* sur l'interface IF1 (voir Figure III.22).

Dans cette approche, des transactions de type *ClkCTr* sont généralement ajoutées au niveau du logiciel de l'application embarquée ou d'un service du système d'exploitation pour effectuer les changements de mode. Chaque mode de puissance d'un CD défini dans la DST et dans la ClkST est donc associé à une activité fonctionnelle qui se traduit par une ou des transactions fonctionnelles pour réaliser ou activer des traitements ou des transferts avec les blocs de ce CD. Dans le cas d'un changement de mode du CD, les transactions fonctionnelles doivent être précédées par une transaction *ClkCTr* adéquate suivie par une phase d'attente de la réponse du PMU qui indique la fin de la transition du mode de puissance pour permettre ainsi la reprise du fonctionnement normal du système. Sur la Figure III.21(a), une transaction *ClkCTr* est ainsi insérée au logiciel embarqué afin de demander au PMU de basculer vers le mode de puissance Clk3 (Figure III.21(b)). Cela se fait en écrivant dans le registre d'état du PMU «PM_ClkST_STATUS_REG» du CSR la valeur spécifique 0x3. Comme le montre la Figure III.21(a), cette transaction *ClkCTr* précède la transaction fonctionnelle d'écriture dans le registre «CpA_START_REG» relatif au bloc CpA. Pour cela, le bloc CpA doit être actif avant la réception de cette transaction fonctionnelle. La ligne de code cpu_relax() est ainsi ajoutée au code du logiciel embarqué afin de permettre au PMU de traiter la transaction *ClkCTr* et ensuite à l'initiateur de l'architecture (le CPU) de ce changement de mode de reprendre le contrôle via l'ordonnanceur SystemC. La fonction cpu_relax() contient essentiellement un *wait* SystemC [sys].

Nous expliquons brièvement dans la suite le fonctionnement interne du PMU. Lors de la réception d'une transaction *ClkCTr*, le PM examine la demande. Deux cas peuvent se présenter :

– **Demande de changement de mode au niveau des CM :**
Suite à cette demande, le processus interne du PM compare les modes globaux de puissance des CD courants par rapport aux modes de puissance demandés. À l'issue de cette

comparaison, le PM lance des requêtes vers les CM concernés par un changement de mode. Cette requête regroupe les facteurs de division des DE d'un CD. À la réception de cette requête, le CM compare le mode local de puissance pour chaque DE par rapport au mode de puissance cible. Cela est fait en comparant les facteurs de division. Dans le cas où les modes courant et cible sont différents, le CM bascule le DE vers le mode cible en générant une nouvelle fréquence avec le nouveau facteur de division. Conséquemment, une mise à jour de la consommation d'énergie locale au niveau DE se déclenche suivie d'une mise à jour globale au niveau du CD en utilisant les fonctions prédéfinies dans *ClkARCH* par l'intermédiaire des moniteurs associés aux DE et au CD. Lorsque le CM termine son traitement de demande de changement de mode, il renvoie un acquittement au PM à travers l'interface IF2 afin de l'informer de la fin du traitement. Le PM reste ainsi en attente de réponse de toutes les requêtes envoyées aux CM du PMU avant de transmettre à son tour un acquittement à l'initiateur de la demande afin de poursuivre la simulation SystemC-TLM.

– **Demande de changement de mode au niveau du *DPLL_Controller* :** Selon le mode demandé, deux scénarios de gestion de puissance peuvent se présenter :

1. Si le mode de puissance demandé est un mode de dérivation (utilisation d'horloges de *bypass*) : le PM envoie directement une requête vers le *DPLL_Controller* demandant l'application de ce mode. Suite à la réception de cette demande, le *DPLL_Controller* bascule toutes les DPLL concernés vers le mode de dérivation. Ainsi, les CM reçoivent des nouvelles sources d'horloge ce qui engendre une mise à jour des horloges internes (les horloges générées à destination des entrées des DE) au niveau du CD. Chaque CM termine son opération de mise à jour et informe le *DPLL_Controller*. Étant donné que le *DPLL_Controller* gère les DPLL inclus dans le même CD de type *container*, il reste en attente des acquittements de tous les CM de ce CD. Dès qu'il les reçoit, il informe le PM à travers l'interface IF22 en indiquant que le mode demandé a été défini avec succès.

2. Si le mode de puissance demandé est un mode de puissance correspondant à une ligne du DST : le PM transmet la requête au *DPLL_Controller* afin d'appliquer le mode demandé. Suite à cette requête, le *DPLL_Controller* récupère le mode courant. Ensuite, il bascule dans un premier temps vers le mode de dérivation en suivant le scénario décrit précédemment. D'un point de vue pratique, cela permet aux DPLL de verrouiller la phase et la fréquence cible par rapport à l'horloge de référence. Ici, nous ne considérons qu'un temps de simulation pour représenter cette opération. Le

DPLL_Controller compare alors le mode demandé avec le mode courant. Puis, il communique aux DPLL concernés à travers l'interface IF33 les nouveaux facteurs de division et de multiplication. En appliquant ces facteurs, les nouvelles sources d'horloge sont générées et le *DPLL_Controller* fixe à nouveau l'horloge de référence comme entrée des DPLL. Cela conduit à une mise à jour de tout l'arbre d'horloge. Ainsi, les CM à leur tour mettent à jour leurs horloges de sortie. Le *DPLL_Controller* reçoit alors des acquittements des CM ce qui lui permet enfin de signaler au PM à travers l'interface IF22 le changement effectif de mode. Suite à ce signal, le PM transmet à l'initiateur un acquittement de la demande afin de poursuivre la simulation SystemC-TLM.

3.2.3 Étape de la simulation complète

A cette étape, l'architecture complète y compris son comportement fonctionnel et sa gestion de puissance est simulée. Les mises à jour des valeurs de consommation de puissance sont tracées dans des fichiers journaux générés automatiquement pendant la simulation. Ces fichiers sont utiles pour ensuite analyser et comparer les différentes solutions de gestion de puissance (par exemple, différentes configurations des facteurs de division/multiplication dans les tables ClkST et DST) ainsi que pour sélectionner la solution de gestion de puissance qui offre un bon compromis entre la performance du système, la consommation de puissance dynamique et la complexité du système de gestion de cette puissance consommée (nombre de CD, taille des tables, diversité des fréquences considérées par exemple).

Bien que le comportement fonctionnel de la plateforme SystemC-TLM soit supposé correct avant l'application de notre méthodologie, il est indispensable de vérifier que l'ajout des concepts permettant la gestion de la distribution d'horloge ne provoque pas des erreurs ou des incohérences. Afin de vérifier que les étapes de la méthodologie se sont déroulées correctement, un processus de vérification a été ajouté au flot de conception de niveau système proposé.

3.2.4 Étape orthogonale de vérification de la gestion d'horloge

L'objectif de cette étape est de vérifier des propriétés de gestion d'horloge au cours de la simulation. Notre solution de vérification se base sur la notion de contrat [Mey92]. Un contrat est défini comme un ensemble de clauses *assume* (i.e. préconditions) et *guarantee* (i.e. post-conditions). Pour vérifier un contrat en simulation, chaque clause est introduite dans *ClkARCH* sous la forme d'une assertion spécifique. Cette assertion est appelée au niveau du code fonctionnel SystemC-TLM afin de déclencher une exception lors de la violation d'un contrat. Il

s'agit donc de spécifier sous la forme de contrats les propriétés de gestion d'horloge à vérifier lorsque les composants interagissent. Pour définir ces contrats, nous avons distingué deux types de composants : des composants fonctionnels et des composants orientés *clock*.

Pour détecter des erreurs liées à une mauvaise utilisation des concepts introduits sur la modélisation d'un arbre d'horloge ou à un mauvais comportement du CM, deux catégories de contrats ont été définies dans notre méthodologie. Leur but est de limiter les erreurs de spécification en vérifiant les propriétés structurelles et/ou fonctionnelles qui reposent sur les interactions entre ces deux types de composants. Ces contrats sont vérifiés durant l'application de notre méthodologie (Figure III.20). Nous donnons ci-dessous les principaux objectifs de vérification de chaque type de contrat et nous illustrons leur intérêt par des exemples.

1. **Contrats de type 1 :**
   Ils sont principalement utilisés lors de la phase de spécification du *clock intent*. Ces contrats vérifient que l'architecture de la distribution d'horloge est correctement structurée et que les règles de composition sont respectées. Les contrats de ce type considèrent les composants orientés *clock*. Il s'agit par exemple de vérifier que :
   - Un *Clock Domain* est valide : il ne contient qu'une seule DPLL et un seul CM au plus.
   - Le nombre d'entrées (colonnes) dans la table ClkST est égal au nombre de *Design_Element* définis dans l'architecture.
   - Un *Design_Element* est inclus dans un seul *Clock Domain* valide.
   - Un changement de valeurs des facteurs N et M d'une DPLL est précédé soit par la sélection de l'horloge de dérivation à destination du CM soit par une mise en mode *clock gated* de l'ensemble du CD.

2. **Contrats de type 2 :**
   Utilisés en phase de simulation, ces contrats permettent de vérifier que les changements de mode de puissance sont cohérents. Nous citons les exemples de contrats suivants :
   - Il n'y a pas d'activité fonctionnelle dans un *Clock Domain* dont l'état serait *clock gated*.
   - Un *Design_Element* ne doit pas recevoir une transaction fonctionnelle s'il est en mode *clock gated*.

D'autres contrats de type 2 ont été définis qui mettent en relation des *Clock Domains* et les *Power Domains*. Ceux-ci sont présentés dans le chapitre IV.

4 Application sur un cas d'étude

Dans cette section, nous présentons la mise en œuvre de notre méthodologie et sa validation par des expérimentations. Nous décrivons notre cas d'étude qui présente une plateforme virtuelle implémentée en SystemC-TLM natif. L'application de notre méthodologie sur ce cas d'étude est détaillée afin d'évaluer les apports des implémentations réalisées.

4.1 Architecture de la plateforme SystemC-TLM

Pour évaluer notre approche, nous avons développé une plateforme virtuelle SystemC-TLM approximativement temporisée (AT). Cette plateforme a été définie afin de révéler les différents états fonctionnels qui peuvent ensuite être utilisés pour des optimisations en puissance/énergie. L'application embarquée de cette plateforme implémente des algorithmes d'encodage et décodage pour des fichiers audio de parole. Dans cet exemple, des fonctions de quantification et de codage/décodage issues des normes ITU G711 et ITU G726 sont utilisées. La Figure III.23 présente l'architecture de la plateforme nommée Audio. Dans une architecture réaliste, différents composants fonctionnels auraient sans doute des réalisations différentes, en particulier sous forme logicielle. Cependant, pour faire apparaître des possibilités de partitionnement en *Clock Domains*, nous avons choisi d'utiliser des modèles de blocs matériels pour ces fonctions. Notre plateforme Audio est constituée des blocs suivants :

- Un CPU a le rôle d'organiser essentiellement les transferts de données entre les autres composants. Ici, le CPU est modélisé par une exécution native du logiciel de l'application.
- Un contrôleur SRAM gère l'interface avec une SRAM où sont rangés les programmes et les données accédés par le CPU.
- Un Timer génère des interruptions périodiques afin de déclencher l'activité des autres composants.
- Un contrôleur d'interruptions reçoit les signaux d'interruption depuis les composants fonctionnels.
- Un CNA représente l'interface vers un haut-parleur et le CAN celle avec un microphone. CAN et CNA contiennent des buffers d'échantillons discrétisés.
- G711 :ENC et G711 :DEC effectuent respectivement l'encodage et le décodage suivant la loi A ou la loi μ définies dans la norme ITU G711 des échantillons issus du CAN ou à destination du CNA. Les modèles de ces composants utilisent une exécution native des fonctions associées.
- G726 :ENC et G726 :DEC effectuent une compression/décompression de type ADPCM

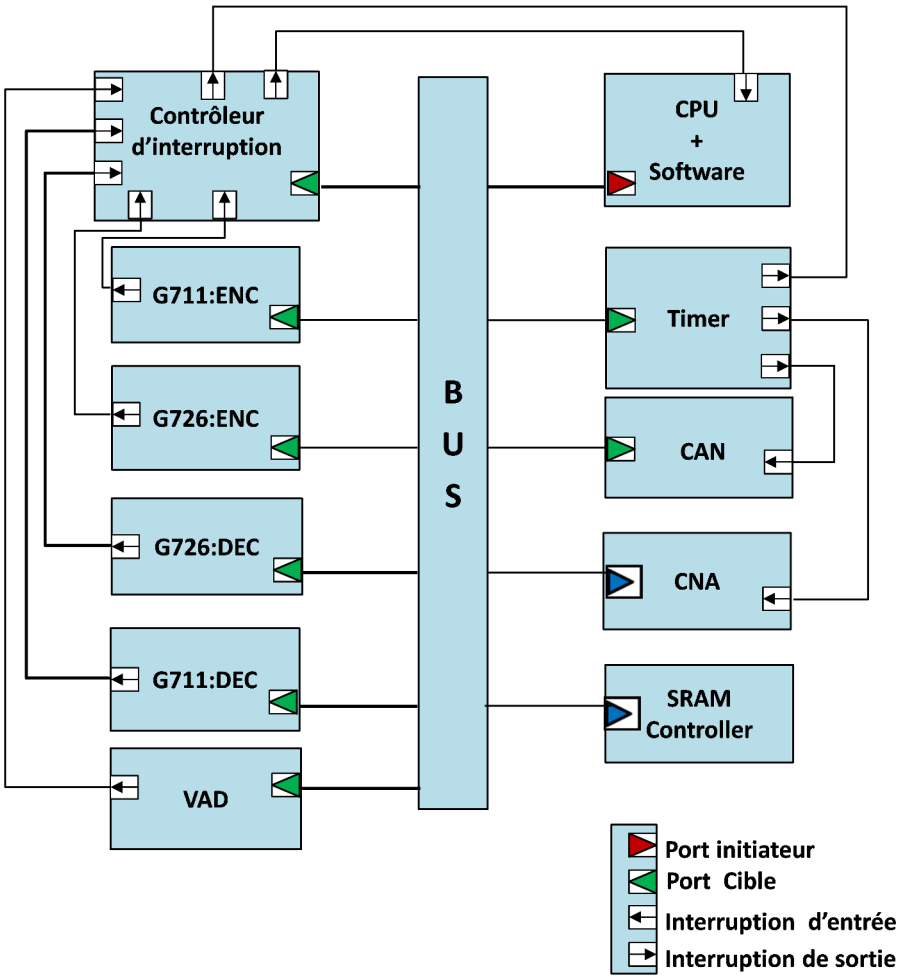


Figure III.23 – Architecture de la plateforme Audio

sur les échantillons issus de G711 :ENC et G711 :DEC et suivant des débits de 40, 32, 24 ou 16 kbit/s définis dans la norme ITU G726. Les modèles de ces composants utilisent une exécution native des fonctions associées.

– Un composant VAD (*Voice Activity Detection*) détecte si un signal de parole est présent. Ce composant est également modélisé sous forme native.

Dans cette architecture, le CPU est le seul initiateur sur le bus. Tous les autres blocs sont des cibles sur le bus. Des signaux d'interruption entre les blocs sont modélisés afin d'assurer les fonctionnalités attendues. Cette plateforme Audio exécute trois différents scénarios applicatifs : *Audio Player*, *Audio Recorder* et *Audio Player/Recorder* qui est une mise en séquence des deux

scénarios précédents.

Avant de décrire ces scénarios, nous précisons que les traitements s'effectuent sur des buffers de k échantillons. Ces buffers sont présents en entrée de chaque composant G711 :DEC, G726 :DEC, G711 :ENC, G726 :ENC, VAD, CNA. Les transferts entre les buffers des différents composants sont effectués par le CPU qui est averti par interruption qu'un composant a terminé son traitement en vue de passer au composant suivant dans la chaîne de traitement.

Par ailleurs, comme la fréquence d'échantillonnage du signal de parole est fixée à 8 kHz, un bloc de k échantillons est traité suivant toute la chaîne de traitement du scénario *Audio Player* ou *Audio Recorder* toutes les $k * 125\ \mu s$.

Lors de l'exécution du scénario *Audio Player*, un flux audio est décodé/décompressé mettant en jeu les composants suivants : SRAM, G711 :DEC, G726 :DEC, CNA, CPU, Timer, le contrôleur d'interruption. La chaine de traitement de ce scénario est organisée comme suit : A chaque $k * 125\ \mu s$, une interruption déclenche la lecture d'un *Bit Stream* de k échantillons. Si le *Bit Stream* indique qu'il s'agit d'un silence (suivant le test effectué par le VAD) des échantillons correspondant à un bruit sont générés en remplacement des échantillons correspondants au silence. Dans le cas contraire, le CPU copie un buffer de k échantillons de la mémoire vers G726 :DEC. Puis, un décodage G726 est effectué suivi d'un décodage G711. Durant l'exécution de ce scénario, le CNA est activé par une interruption du Timer toutes les 125 μs afin d'écrire un échantillon décodé dans le fichier résultat.

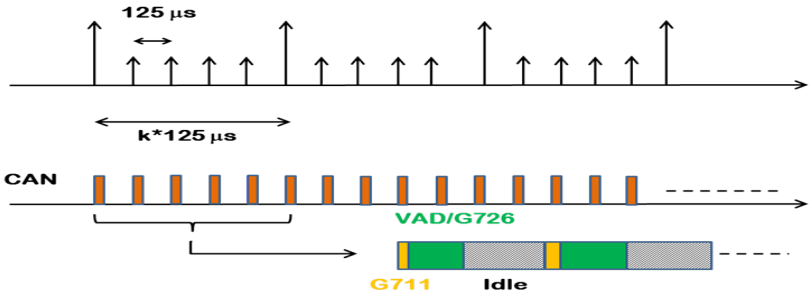


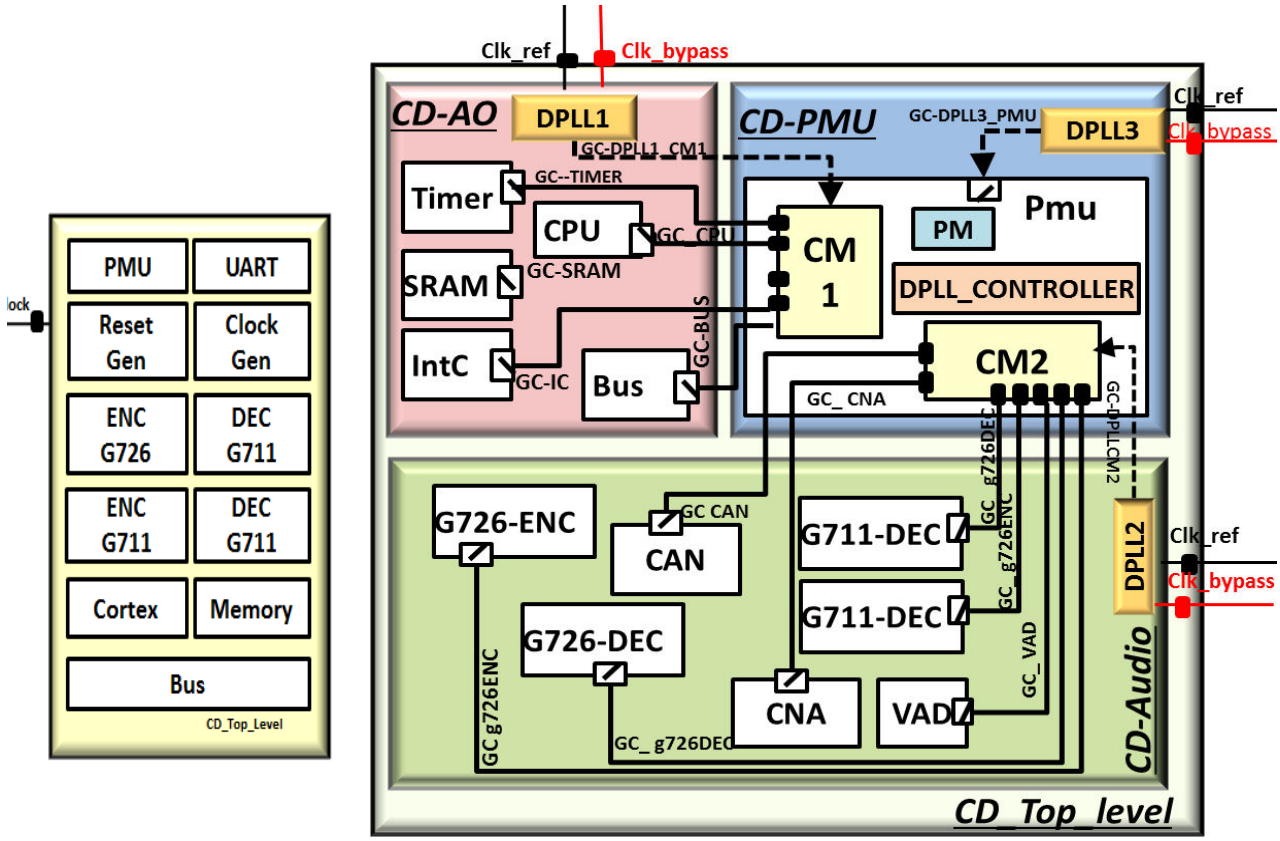
Figure III.24 – Exemple de traitement sur le scénario *Audio Recorder*

Dans la suite, nous décrivons le scénario *Audio Recorder* qui correspond aux modes de puissance de la ClkST et de la DST (voir Figure III.26) selon la décomposition en *Clock Domains* représentée sur la Figure III.25(b). Lors de l'application de ce scénario, un flux audio est encodé/compressé mettant en jeu les composants SRAM, G711 :ENC, G726 :ENC, VAD, CAN, CPU, Timer et le contrôleur d'interruption. L'organisation des traitements est montrée dans la Figure III.24 dans le cas du scénario *Audio Recorder* : La simulation démarre en mettant

tous les CD de la Figure III.25(b) en mode de dérivation. Durant ce mode, les périphériques sont initialisés et les DPLL sont en phase de verrouillage jusqu'à ce qu'elles soient prêtes à générer l'horloge (GC) aux différents CM. Une fois cette étape d'initialisation effectuée, toutes les 125 μs, le CAN est activé par une interruption du Timer pour lire un échantillon du flux audio (*Read_from_file state*). Toutes les $k * 125$ μs, un buffer de k échantillons est alors traité. Tout d'abord, ce buffer est examiné par le VAD (*Voice detection state*). Si un signal de parole est détecté, un encodage G711 est effectué (*G711 Enc state*) suivi d'un encodage G726 (*G726 Enc state*). Ensuite, les échantillons encodés sont sauvegardés dans la mémoire (SRAM). En absence d'un signal de parole, l'encodage G711 et l'encodage G726 ne s'exécutent pas et un *flag* dans le *Bitstream* indiquant le début d'un silence est enregistré dans la SRAM. Les composants restent en attente du prochain buffer (*Read_from_file state*). Le temps de traitement est a priori inférieur à la période de $k * 125$ μs ainsi les composants correspondants sont dans un état *Idle* pendant une partie de la période de $k * 125$ μs. Suivant le taux de compression appliqué pour G726, le nombre d'échantillons k peut varier en considérant une taille de buffer constante aussi le temps où les composants sont *Idle* entre deux traitements de buffer peut varier en fonction de k. La fréquence d'horloge de ce composant peut être modifiée lorsqu'on connait le taux de compression appliqué (et tant que la latence de bout en bout maximum autorisée n'est pas dépassée). Comme le montre la Figure III.26, durant l'exécution du scénario *Audio Recoder* les composants G711 :DEC, G726 :DEC et CNA sont inactifs. Un contrôle du *Clock Domain* par *clock gating* est alors appliqué sur ces composants.

4.2 Évaluation de la consommation d'énergie dynamique

Les caractéristiques de puissance dépendante de la technologie considérée pour cette architecture (telles que la capacité de commutation par composant et les courants de fuite) sont extrapolées dans notre cas de celles issues de la spécification du circuit *Ultra-low-power 32-bit MCU ARM* [LP]. Ces caractéristiques sont attachées aux différents DE de la plateforme afin d'évaluer la consommation d'énergie à chaque changement d'état activé par le PMU. Les résultats illustrés sur la Figure III.27 représentent la consommation d'énergie résultant de l'encodage d'un signal durant 7 secondes de paroles selon le scénario *Audio Recorder*. Ce scénario est exécuté sur la plateforme augmentée par le *clock intent* de la Figure III.25(b). Dans certains intervalles de temps, nous remarquons (Figure III.27) que le *CD-Audio* consomme une faible énergie. Ces intervalles correspondent au traitement des périodes de silence. Durant ces



(a) Plateforme Audio sans appliquer du *clock intent*

(b) Plateforme Audio en appliquant un *clock intent*

Figure III.25 – Plateforme Audio avec et sans *clock intent*

Div. Factors / GCS	GC Timer	GC Bus	GC CPU	GC IntC	GC Sram	GC g726ENC	GC g726DEC	GC g711ENC	GC g711DEC	GC CAN	GC CNA	GC VAD
Read_from_file	8	2	Off	Off	2	Off	Off	Off	Off	4	Off	Off
Voice detection	8	2	1	8	2	Off	Off	Off	Off	4	Off	2
G711 Enc	8	2	1	8	2	Off	Off	4	Off	4	Off	Off
G726 Enc	8	2	1	8	2	1	Off	Off	Off	4	Off	Off

(a) Clock State Table (ClkST)

Div. & Mul Factors / GCS	DPLL_AO		DPLL_ENC_DEC	
	M	N	M	N
High	5	2	4	1
Low	1	2	1	1

(b) DPLL State Table (DST)

Figure III.26 – ClkST et DST dans le cas de scénario *Audio Recoder*

périodes, le composant CAN du *CD-Audio* est actif tandis que les autres sont dans l'état *clock gated* tel que défini dans la ClkST par utilisation du mode de puissance *Read_from_file state*. Comme nous pouvons le voir également sur la Figure III.27, la consommation d'énergie de CD : *CD-PMU* est constante pendant toute la durée de l'exécution. Cela est dû au fait que le PMU est alimenté directement par sa DPLL fonctionnant à une fréquence faible et stable. Le *CD_TOP* de la Figure III.27 représente l'ensemble du système et donc la puissance consommée par l'ensemble des CD.

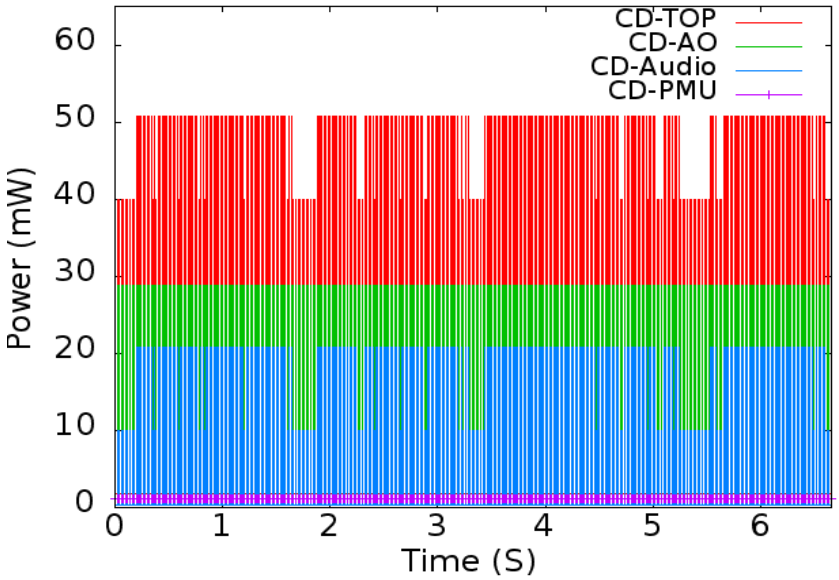
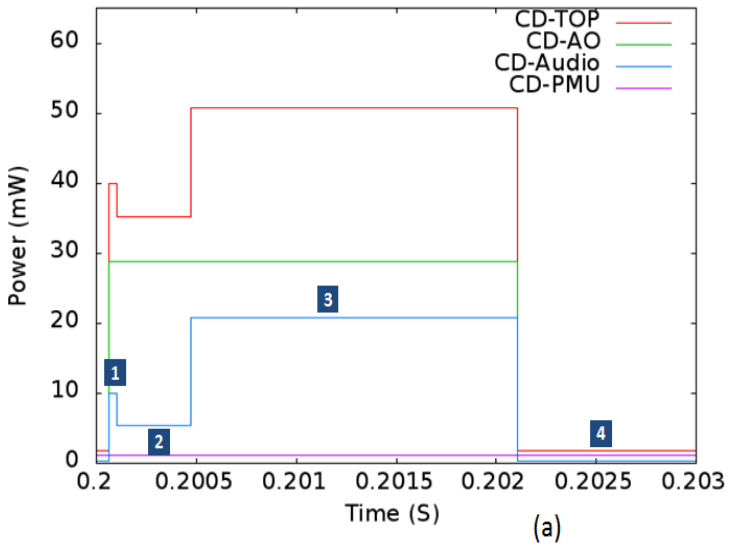


Figure III.27 – Puissance dynamique consommée pendant l'exécution du scénario *Audio Recorder* sur la plateforme Audio augmentée par un *clock intent*

Afin d'observer précisément l'impact de notre méthodologie sur la distribution d'horloge au cours de la simulation, nous faisons un zoom sur des modes de puissance successifs durant l'encodage d'un buffer de 160 échantillons. Cette exécution suit la spécification du *clock intent* présentée sur la Figure III.28. La zone 1 de la Figure III.28 représente l'étape de détection de signal de parole (*voice detection state*). Comme le buffer d'échantillons contient un signal de parole, un encodage est effectué par G711 (zone 2 de la Figure III.28). Puis, un encodage G726 est effectué suivi par une sauvegarde dans la mémoire (zone 3 de la Figure III.28). Durant cette étape, le CD : *CD-Audio* a la plus forte consommation d'énergie dynamique. Ceci est le résultat de l'application du mode de puissance *G726 Enc state*. En fait, durant ce mode le composant G726 :Enc utilise une fréquence plus élevée par rapport autres composants. La zone 4 présente la phase où le CPU est en attente du prochain buffer à traiter ce qui explique l'inactivité du

CD *CD-Audio*.



IP \ Step	Frequency (MHZ) 1	2	3	4
VAD	64	0	0	0
G711:ENC	0	32	0	0
G726:ENC	0	0	128	0
CPU	64	64	64	0

(b)

Figure III.28 – Zoom sur une succession des modes de puissance

Afin de montrer les gains énergétiques obtenus en appliquant notre méthodologie, nous avons comparé la consommation d'énergie de l'exécution de notre plateforme sans aucun contrôle et sans gestion d'arbre d'horloge (Figure III.25(a)) avec la consommation d'énergie produite à l'exécution de la plateforme augmentée par le *clock intent* (Figure III.25(b)). Les résultats de simulation montrent l'intérêt évident de partitionner une plateforme en différents *Clock Domains* puisque l'alternative avec *clock intent* fournit un gain énergétique de 70% par rapport à la plateforme non partitionnée. Notons que pour obtenir les résultats de simulation de la plateforme de la Figure III.25(a) nous pouvons utiliser la plateforme de la Figure III.25(b) en initialisant par le CPU via le PMU les horloges suivant la configuration de la Figure III.25(a) et en n'appliquant aucune requête par le logiciel de changement de mode de puissance. Ceci illustre qu'avec la même architecture, et uniquement par utilisation des différents paramètres de contrôles des modèles permettant de structurer l'architecture en *Clock Domains* il est aisé d'explorer de différentes configurations de *clock intent*.

L'approche proposée dans ce chapitre pour structurer une plateforme SystemC-TLM suivant

un *clock intent* a été considérée également sur cet exemple de plateforme Audio en utilisant l'outil commercial *Platform Architect MCO* de *Synopsys*. Cet outil de prototypage est basé sur SystemC-TLM. Pour ce faire, les différents modèles des composants nécessaires à la structuration de l'arbre d'horloge au niveau système ont été introduits dans cet outil afin d'obtenir une plateforme opérationnelle contrôlée en puissance dynamique. L'intérêt de cette étude était d'évaluer la possibilité d'utiliser *ClkARCH* dans un environnement commercial de simulation. Le détail de ce travail est présenté en Annexe A.

Ce cas d'étude montre l'intérêt de notre approche pour concevoir et vérifier tôt dans le flot de conception une architecture dont nous souhaitons optimiser sa consommation en puissance dynamique. La vitesse de simulation inhérente du niveau système permet également d'explorer rapidement plusieurs alternatives d'architecture avec différentes décomposition en CD et d'évaluer pour chacune d'elles le gain en énergie. Il faut noter que quel que soit le type de plateforme, l'ajout du *clock intent* et des contrats n'entraine qu'une augmentation négligeable des temps de simulation (environ 2.6%). Ceci est principalement lié au fait que par rapport à l'ensemble de l'activité fonctionnelle de l'architecture, les événements relatifs à la gestion des *Clock Domains* sont finalement rares et n'entrainent donc qu'un faible surcoût en temps de simulation.

5 Conclusion

Nous avons présenté dans ce chapitre notre environnement *ClkARCH* pour décrire un *clock intent* au niveau transactionnel. Nous avons décrit aussi notre méthodologie d'insertion de stratégies de gestion d'horloge dans une plateforme SystemC-TLM où chaque étape s'appuie sur l'environnement *ClkARCH*. Notre méthodologie permet ainsi de modéliser un arbre d'horloge d'une plateforme fonctionnelle décrite au niveau système et de vérifier la cohérence des décisions prises par le PMU vis-à-vis des états induits par les comportements fonctionnels. Cette cohérence est automatiquement vérifiée en simulation suivant les propriétés définies et exprimées via l'insertion de contrats. En effet, notre méthodologie permet de modéliser un *clock intent* en interaction avec la modélisation fonctionnelle pour une validation conjointe des performances d'un système et de la stratégie de gestion de puissance. Cette stratégie de gestion de puissance impacte directement les performances puisqu'elle contrôle les fréquences d'horloge des composants. De plus, elle permet de comparer plusieurs stratégies de réduction de puissance ainsi que différentes architectures en termes de décomposition en CD. Même si notre approche s'inscrit dans une démarche de séparation des préoccupations concernant le modèle fonctionnel et le modèle du *clock intent*, cela implique la création de différentes versions de

la fonction *sc_main()* du modèle fonctionnel initial étendue avec chaque implémentation de stratégie de gestion de puissance. Afin d'évaluer rapidement les différentes solutions possibles, nous proposons dans le chapitre suivant une approche structurelle pour produire des modèles de simulation SystemC-TLM d'architectures matérielles intégrant la description de système de gestion de puissance.

Chapitre IV

Une approche structurelle pour produire des modèles de simulation SystemC-TLM d'architectures matérielles intégrant la description de système de gestion de puissance

1 Introduction

Le standard IP-XACT [IP-] permet une représentation unifiée de la structure d'une architecture matérielle dans le but de garantir la consistance des informations à chaque niveau du flot de conception. Il favorise aussi la maîtrise de la complexité de la description de plateformes matérielles en facilitant la réutilisation et l'intégration des blocs IP. Si dans un premier temps IP-XACT a été développé pour cibler le niveau RTL, il a ensuite évolué pour être un support à la modélisation au niveau TLM. Il permet alors de monter dans le niveau d'abstraction et de profiter de tous les avantages de la modélisation au niveau transactionnel. Ce standard fournit également un mécanisme d'extension simple à utiliser qui permet d'ajouter des informations non prévues par la norme et donnant ainsi la possibilité aux concepteurs d'IP de représenter de manière structurée dans un standard leurs composants personnalisés. En outre, IP-XACT offre un mécanisme d'automatisation de flots de conception grâce aux éléments *generator* et *generator chain*.

Cependant, dans sa version actuelle IP-XACT ne définit aucun aspect permettant de décrire une structure de gestion de puissance. Notre objectif est d'étudier comment utiliser IP-XACT et ses mécanismes d'extension pour décrire une plateforme qui combine l'aspect fonctionnel et l'aspect non fonctionnel (gestion *Power Domain* + *Clock Domain*). De plus, le but est d'extraire et de traiter ces informations pour produire un modèle SystemC-TLM/C++ de

description d'une structure relative à des stratégies de gestion de puissance adaptées au modèle fonctionnel.

2 Spécification d'une plateforme fonctionnelle SystemC-TLM en IP-XACT

Cette section décrit les étapes d'intégration d'une architecture décrite en SystemC-TLM ainsi que sa structuration selon notre stratégie de gestion de puissance dans l'environnement industriel Magillem. Des outils et des environnements industriels comme *Magillem* [Mag], *Nauet* [Nau], *plugin Eclipse* développé par ARM [ecl], *Scarlet* [Sca] et *Duolog* [Duo] ont été développés afin de faciliter l'utilisation du standard IP-XACT, en fournissant des fonctionnalités pour l'automatisation de flots d'outils de conception et l'intégration de composants matériels. Parmi ces environnements IP-XACT, nous avons choisi d'utiliser l'outil Magillem pour différents raisons. En effet, Magillem est un outil utilisé pour décrire en IP-XACT l'architecture et les caractéristiques d'une plateforme aux niveaux RTL ou TLM. Il propose une interface schématique et permet notamment la génération d'architectures dans différents langages (vhdl, verilog, SystemC). L'ensemble des fonctionnalités du standard IP-XACT nécessaire à notre étude est implémenté dans cet outil. En outre, le travail décrit dans cette thèse est réalisé dans le cadre du projet collaboratif HOPE (*Hierarchically Organized Power/Energy management*) en partenariat avec MDS (*Magillem Design Services*). Cela nous a permis d'avoir l'accès à l'ensemble des différentes fonctionnalités de l'outil.

2.1 Présentation de l'environnement de développement Magillem

L'environnement Magillem est un IDE qui fournit des services centrés sur le standard IP-XACT. Il est proposé par l'entreprise MDS (*Magillem Design Services)* [Mag] et il est composé de plusieurs modules (Figure IV.1) :

- **MIP (*Magillem IP-XACT Packager*) :** Cet outil permet la représentation en IP-XACT des composants matériels, pour les niveaux RTL ou TLM. Il intègre des éléments d'importation automatique à partir de modèles RTL ou de fichiers texte de type "csv". Il propose également une interface graphique permettant d'entrer manuellement les informations de description d'un bloc matériel en IP-XACT. Les fonctionnalités de ce module sont également accessibles en ligne de commande ou via l'interface TGI définie dans IP-XACT.

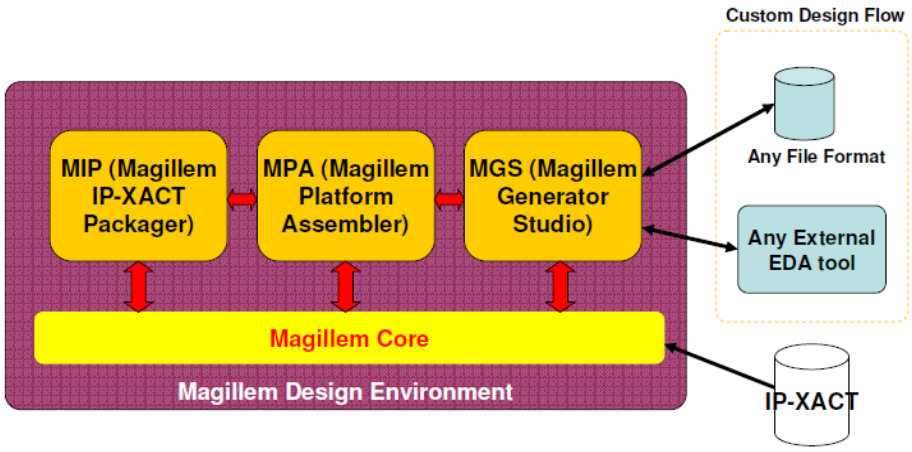


Figure IV.1 – Environnement Magillem [Mag]

- **MPA (*Magillem Platform Assembly*) :** Cet outil est utilisé pour décrire en IP-XACT les caractéristiques d'une plateforme aux niveaux RTL ou TLM (instanciation et configuration des composants matériels, connexions de type ad-hoc ou de type interface, gestion des configurations). Il propose une interface schématique et permet notamment la génération des architectures dans différents langages (vhdl, verilog et SystemC). Les fonctionnalités de ce module sont également accessibles en ligne de commande ou via l'interface TGI définie dans IP-XACT.
- **MGS (*Magillem Generator Studio*) :** Ce module apporte toutes les fonctionnalités nécessaires au développement et au débogage des moteurs utilisant l'interface TGI de IP-XACT. Cela permet de programmer les différentes étapes qui interviennent dans un flot de conception et de vérification.

2.2 Modélisation de la plateforme Audio en IP-XACT dans l'environnement Magillem

Notre objectif est de modéliser en IP-XACT une architecture décrite en SystemC-TLM. Pour ce faire nous suivons le flot de conception IP-XACT présenté dans la section 2.4. Nous utilisons la plateforme Audio (Figure III.23) pour illustrer la démarche suivie.

Dans la suite, nous détaillons la construction d'une bibliothèque de composants IP de la plateforme Audio dans l'environnement Magillem. Ensuite, nous décrivons l'importation et l'assemblage des instances des composants dans un élément *Design* correspondant à la plateforme.

2.2.1 Création des composants

Avant de créer les composants, il est nécessaire de définir les interfaces de communication. Pour atteindre cet objectif, la première étape consiste à créer l'élément *BusDefinition* qui définit, tel qu'indiqué dans le chapitre II, les aspects haut niveau d'un bus. Nous avons fixé le nombre maximal de *masters* pouvant être connectés sur le bus à 1 puisque notre architecture cible vérifie cette propriété. Toutefois, cette valeur peut être modifiée à tout moment. Également, nous avons spécifié à ce niveau que les bus définis ultérieurement avec ce type pourront contenir des informations d'adressage et de *memory maps*.

Une fois ces éléments d'interface définis, nous sommes passés à la création des différents composants de la plateforme cible en IP-XACT. Pour chaque composant, nous définissons un élément *model* où nous définissons une vue TLM et les ports physiques associés au composant. En effet, une vue en IP-XACT spécifie un niveau de description du composant. Ce dernier peut avoir plusieurs vues ; une vue RTL par exemple pour référencer les fichiers source module/entity, une vue *software* pour référencer le code source de la partie logicielle (le pilote de l'IP par exemple) et une vue pour la documentation.

Dans notre cas, nous avons défini une vue TLM où nous spécifions que le langage de description de l'IP est SystemC-TLM. Nous avons également défini une référence vers les fichiers *header* de chaque composant de l'architecture (Figure IV.2). Par ailleurs, nous avons défini les

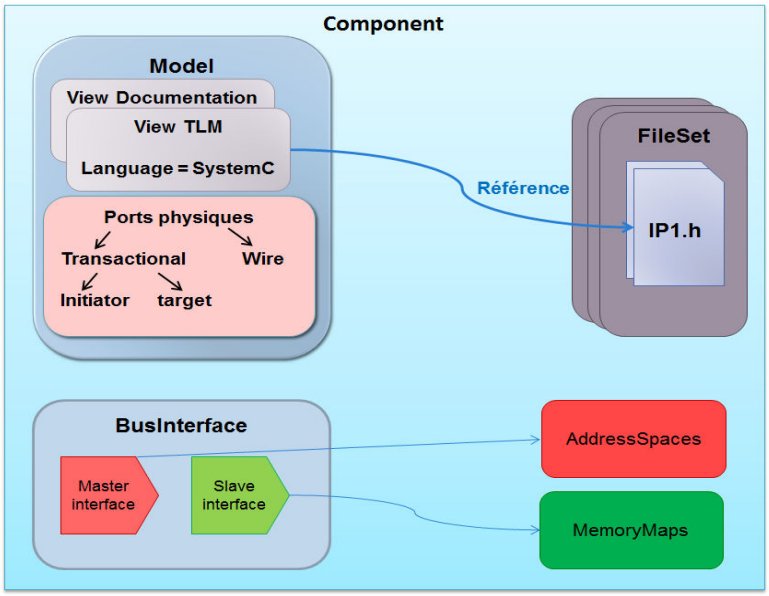


Figure IV.2 – Architecture interne d'un composant

ports physiques de types *transactional* et *wire* selon le type de connexion utilisé. Les ports de types *transactional* transportent des informations au niveau TLM alors que les ports de type *wire* transportent des valeurs exprimées sous forme de vecteurs de bits (données et adresses) mais ne peuvent pas combiner ces informations avec un signal *clock* ou un *reset*. C'est l'élément IP-XACT dit *qualifier* qui spécifie la nature des informations circulant à travers les ports. Les ports transactionnels que nous définissons ici sont les sockets du standard TLM 2.0 [sys].

Ayant tous les ports d'un composant bien représentés, nous avons créé les *BusInterfaces* qui référencent un *BusDefinition* et un *AbstractionDefinition* et spécifient les interfaces vers un composant. Ces éléments nous permettent de définir les interfaces des entités *Master* et *Slave* associées à l'IP, ils permettent également de définir le *PortMap*. En IP-XACT, le *PortMap* décrit le *mapping* entre les ports logiques définis dans l'élément *AbstractionDefinition* et les ports physiques des composants. Puis, pour compléter la description de l'architecture, des détails de granularité plus fine ont dû être ajoutés. Nous avons alors enrichi nos composants créés avec des informations d'adressage. En effet, pour assurer une communication correcte entre le *Master* (CPU de la plateforme Audio) et les esclaves connectés sur le bus (voir l'architecture représentée sur la Figure III.23), il est nécessaire de définir un espace d'adressage (correspondance d'adresses) pour chaque interface *Master* avec un *Slave*.

2.2.2 Création du *Design*

IP-XACT est basé sur une relation hiérarchique entre les éléments *Component* et *Design*. En effet, un composant, tel qu'il est défini par le standard, peut être un composant hiérarchique si nous l'associons à un élément *Design* (Figure II.5). Ainsi, le composant hiérarchique référence un identifiant unique VLNV (Vendeur, Librairie, Nom et Version) du *Design* et éventuellement le VLNV d'un élément *Design Configuration*. Ce dernier permet d'ajouter des informations de configuration supplémentaires d'un *Design* ou d'un *generator chain*. Le composant hiérarchique sera alors une enveloppe du *Design* et contiendra la description de l'architecture de la plateforme cible.

Passons maintenant à l'importation et l'assemblage des instances des composants préalablement définis pour former le *Design*. Nous avons créé un composant hiérarchique auquel nous avons associé notre *Design* où nous avons créé les instances des composants et les connexions entre elles. En effet, IP-XACT définit quatre types de connexions :

- Les ***interconnections*** : Une interconnexion spécifie une connexion entre un *BusInterface* d'un composant et un autre *BusInterface* d'un composant.
- Les ***hierarchical connections*** : Ce type de connexion relie un *BusInterface* de l'instance

d'un composant à celui du composant hiérarchique.

- Les ***ad hoc connections*** : Une *ad hoc Connection* spécifie les connexions entre les ports des instances des composants ou entre les ports d'une instance et ceux du composant hiérarchique.
- Les ***monitor connections*** : Cette connexion est établie entre une interface de moniteur et tout autre mode d'interface : maître, esclave, système, etc. L'interface moniteur est définie pour un seul mode et ne peut être utilisée que pour ce mode. Les connexions moniteurs sont purement utilisées pour l'observation non intrusive d'une interface.

Compte tenu de l'exemple de la plateforme Audio, nous avons choisi d'utiliser les connexions de type *interconnections* pour modéliser les connexions via le bus et les connexions de type *ad hoc connections* pour modéliser les connexions entre les ports.

Pour récapituler, nous avons conçu notre bibliothèque de composants IP en commençant par la définition des interfaces de communication, puis la création des blocs simples que nous avons enrichis par la suite par des informations relatives aux correspondances d'adresses. Ensuite, nous avons créé notre *Design* suivant la plateforme d'étude. À ce stade, nous avons obtenu la description complète de la structure de la plateforme TLM dans sa partie structurelle en IP-XACT. Cette plateforme IP-XACT peut être traduite en une plateforme SystemC-TLM (la partie structurelle du modèle) en utilisant le générateur fourni par Magillem.

3 Augmentation de la plateforme décrite en IP-XACT par la stratégie de gestion de puissance

IP-XACT est un choix pertinent pour la représentation des plateformes pour la flexibilité qu'il offre afin de modéliser les différents détails de l'architecture et surtout pour la possibilité d'ajouter des extensions personnalisables par les concepteurs. C'est cette possibilité d'extension présente dans ce standard qui est exploitée ici pour introduire la description d'une structuration d'une plateforme selon une stratégie de gestion de puissance : stratégie de la gestion d'horloge basée sur *ClkARCH* (Chapitre III) et stratégie de la gestion de puissance basée sur PwArch ([MPA11]).

3.1 Expression d'un *power intent* dans IP-XACT

Magillem a exploité la flexibilité du standard IP-XACT en proposant dans son environnement *Platform Assembly* une extension permettant la description d'un *power intent* au niveau

RTL.

La description des éléments *power* dans une représentation IP-XACT a été réalisée en définissant des commandes TCL qui agissent sur la base de données afin d'y inclure sous forme de *Vendor Extension* les informations et les attributs associés au *power intent* dans les blocs qui composent le *Design*.

Un domaine *top* est défini, il contient l'ensemble des domaines. Ensuite sont décrits les *Voltage Domains* avec un attribut *domaintype* ayant une valeur *Voltage* (Figure IV.3(b)). Chacun des *Power Domains* est associé à son domaine parent qui est ici un *Voltage Domain*. Une fois ces informations renseignées dans la base de données de *Platform Assembly*, il est nécessaire de renseigner pour chaque bloc IP d'un *Design* les attributs permettant d'associer ce bloc à un *Power Domain* (Figure IV.3(a)). Le standard IP-XACT n'incluant pas les concepts nécessaires pour représenter ce type d'association, le choix s'est naturellement porté sur l'utilisation des *Vendor Extensions* proposées par le standard. Ainsi, dans l'exemple de la Figure IV.4 le composant de nom *Timer* est associé au *Power Domain* de nom *PD_AO*.

À partir d'une telle description, des représentations internes dans l'outil *Platform Assembly* sont construites, en particulier la hiérarchie des *Power Domains* qui peut être déduite des attributs renseignés en IP-XACT.

Ces extensions proposées par Magillem permettent aussi de développer des techniques pour insérer automatiquement des cellules de types *isolation cells* et *level shifters* (section 2.6). Ces techniques permettent en particulier de tendre à réduire l'écart entre le niveau de modélisation RTL et le niveau TLM. Dans nos travaux, elles ne sont pas considérées car ces cellules n'ont pas de sens au niveau TLM tout comme celles de type *clock domain crossing* (Chapitre III. 3).

Dans l'approche de représentation des *Power Domains* dans IP-XACT proposée par Magillem certaines informations sont manquantes pour une génération de code C++ selon les modèles développés dans *PwARCH*. C'est pourquoi avec les informations disponibles, il n'est pas possible d'instancier les *Supply Nets* et les *Design_ Elements* avec leur association avec les blocs IP du modèle fonctionnel. Pour le premier cas, nous avons utilisé les noms des *Voltage Domains* pour définir dans la génération de code C++ les noms des *Supply Nets* en entrée des *power switches* des *Power Domains*.

Dans la librairie C++ *PwARCH*, les *Supply Nets* sont définis comme suit :

Supply_Net(string name, string stype, Power_Domain pPower_Domain) ;*

L'attribut *stype* renseigne le type du *Supply Net*. Il prend l'une des valeurs suivantes : *ground_Net*, *primary_Net* ou *switched* (comme défini dans UPF).

Par conséquent, en plus des *Power Domains* et des *Voltage Domains* que nous avons insérés

```
# Place instances into different Power Domains

aiprtl:: SetAttribute -comp "/cpu_inst_0" -attribute "power: domain" -value "PD_NATIVE"
aiprtl:: SetAttribute -comp "/SRAM_Controller_inst_0" -attribute "power: domain" -value "PD_NATIVE"
aiprtl:: SetAttribute -comp "/VAD_inst_0" -attribute "power: domain" -value "PD_VAD"
```

(a)

```
#Create Power Domain hierarchy

aiprtl: :RegisterDomain -domain "top"
aiprtl: :SetDomainAttribute -domain "top" -attribute "domaintype" -value "top"

aiprtl: :RegisterDomain -domain "VDD_NATIVE" -parentdomain "top"
aiprtl: :SetDomainAttribute -domain " VDD_NATIVE " -attribute "domaintype" -value "voltage"

aiprtl: :RegisterDomain -domain "PD_NATIVE" -parentdomain " VDD_NATIVE "
aiprtl: :SetDomainAttribute -domain " PD_NATIVE " -attribute "domaintype" -value "power"

aiprtl: :RegisterDomain -domain "VDD_SOC" -parentdomain "top"
aiprtl: :SetDomainAttribute -domain " VDD_SOC " -attribute "domaintype" -value "voltage"

aiprtl: :RegisterDomain -domain "PD_AO" -parentdomain " VDD_SOC"
aiprtl: :SetDomainAttribute -domain " PD_AO " -attribute "domaintype" -value "power"

aiprtl: :RegisterDomain -domain "PD_VAD" -parentdomain " PD_AO"
aiprtl: :SetDomainAttribute -domain " PD_VAD " -attribute "domaintype" -value "power"
```

(b)

Figure IV.3 – Script TCL pour insérer la description d'une hiérarchie de *Power Domains* dans une description IP-XACT d'un design

```
<spirit:componentInstance>
    <spirit:instanceName>Timer_inst_0</spirit:instanceName>
    <spirit:componentRef spirit:version="1.0" spirit:name="Timer" spirit:library="Hope" spirit:vendor="LEAT"/>
    <spirit:vendorExtensions>
        <mdsAttr:attributes>
                <mdsAttr:group mdsAttr:groupName="power">
                <mdsAttr:attribute mdsAttr:name="domain">
                    <mdsAttr:value mdsAttr:prompt="Root">PD_AO</mdsAttr:value>
                </mdsAttr:attribute>
```

Figure IV.4 – Exemple d'une association d'un bloc IP à un *Power Domain*

à la description du *Design*, il a été nécessaire d'ajouter à la description un attribut *type* qui permet de renseigner le type du *Supply Net*.

La Figure IV.5 montre un exemple de définition de cet attribut pour le *Supply Net VDDVAD*

```
aiprtl:: AddAttribute -comp "/VAD_inst_0" -attribute "power: supplyNet" -value "VDD_VAD" -type "switched"
```

Figure IV.5 – Définition d'un attribut pour renseigner le type du *Supply Net*

de type *switched* basée sur le langage de script TCL. Ainsi, nous obtenons dans la description IP-XACT la définition du *Supply Net* avec son type (Figure IV.6).

Tel que nous l'avons présenté dans le chapitre précédent, notre approche de gestion de puissance

```
<mdsAttr:attribute mdsAttr:name="supplyNet">
    <mdsAttr:value mdsAttr:prompt="Root" mdsAttr:type="switched">VDDVAD</mdsAttr:value>
</mdsAttr:attribute>
```

Figure IV.6 – Code IP-XACT définissant le *Supply Net* et son type

favorise la séparation des préoccupations entre les modèles fonctionnels et non fonctionnels. Pour relier les concepts de la gestion de puissance avec l'architecture fonctionnelle SystemC-TLM, nous avons défini un objet C++ *Design_Element* qui pointe sur le bloc IP SystemC-TLM. Rappelons que le *Design_Element* (DE) est défini par un ensemble de paramètres. Le premier paramètre est le nom avec lequel l'objet *Design_Element* est instancié dans le *sc_Main* C++ (Figure III.19). Les autres paramètres à considérer concernent principalement ceux utilisés pour calculer les consommations de puissance au sein des *Design_Elements* : valeurs du taux de commutation, de capacitance et de tout autre paramètre qui peut être utilisé dans l'évaluation de la puissance consommée. Les valeurs de ces paramètres doivent donc être renseignées au moment de l'instanciation de l'objet *Design_ Element* dans la description C++. Il est donc nécessaire de compléter la représentation IP-XACT avec les informations requises permettant l'instanciation des *Design_Elements* dans le modèle SystemC-TLM.

IP-XACT offre la possibilité d'associer à un composant un ensemble de paramètres dont la configuration sera choisie lors de l'instanciation de ce composant dans le *Design*. Ces paramètres peuvent être encapsulés dans un élément *view* ce qui permet de mieux structurer la description et de renforcer la séparation entre l'aspect fonctionnel et l'aspect non-fonctionnel. D'où, nous avons défini une vue *power* associée à chaque modèle d'un composant en IP-XACT. Dans cette vue *power*, nous avons créé des paramètres pour définir le *Design_Element* et les paramètres qui lui sont associés pour évaluer lors de la simulation la consommation de puissance (Figure IV.7). Notons qu'il est possible de définir des valeurs par défaut dans la description IP-XACT du composant pour ces paramètres configurables.

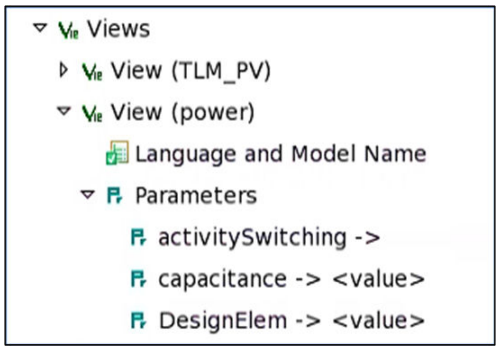


Figure IV.7 – La vue *power* et des exemples de paramètres configurables

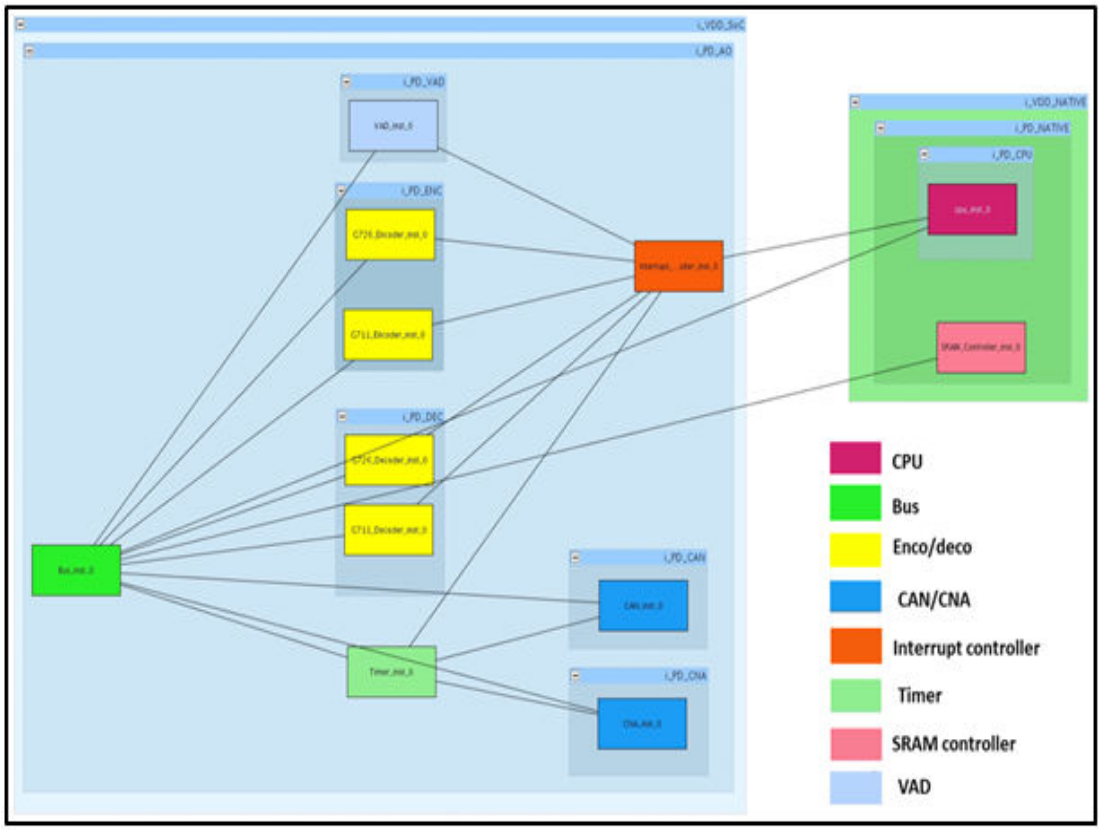


Figure IV.8 – Représentation sous *Platform Assembly* de l'architecture illustrée sur la Figure IV.9

En utilisant cette approche de définition des *Power Domain* et *Voltage Domain*, des *Supply Nets* et de l'association des composants aux *Power Domains*, il est ainsi possible de décrire une structure hiérarchique complète d'un *power intent* suivant l'approche basée sur *PwArch*.

L'architecture de la plateforme Audio donnée dans la Figure IV.8 et saisie dans *Platform As-*

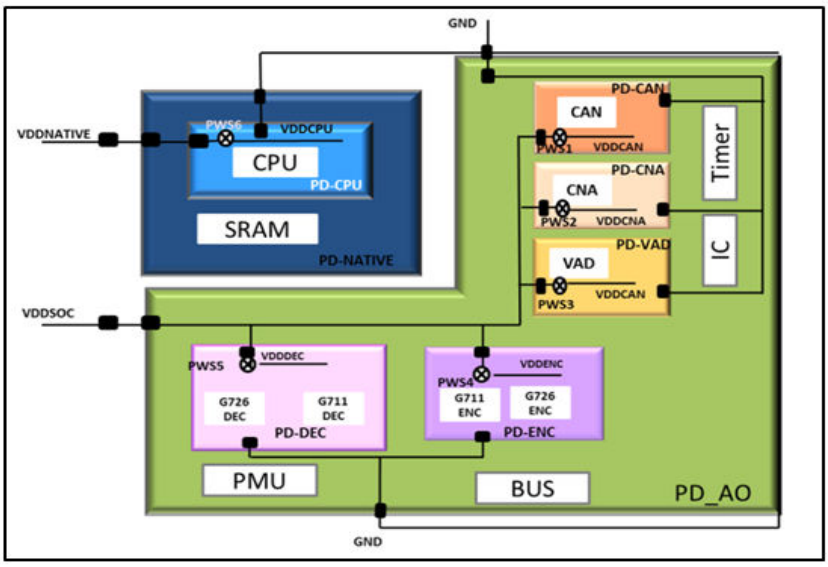


Figure IV.9 – Exemple de la plateforme Audio structurée en *Power Domains* avec indication des *Supply Nets* associés (*Voltage Domain*)

sembly permet d'obtenir une description IP-XACT sur laquelle nous avons appliqué la structure en *Power Domains* illustrée sur la Figure IV.9. La représentation graphique de la Figure IV.8 est construite à partir de la hiérarchie définie par la déclaration des *Power Domains*.

Dans le paragraphe suivant, nous décrivons l'approche proposée pour intégrer un *clock intent* dans une représentation IP-XACT selon la stratégie décrite dans le chapitre III.

3.2 Expression d'un *clock intent* dans IP-XACT

Dans IP-XACT, la notion d'horloge est présente mais dans un niveau de modélisation RTL c'est-à-dire trop détaillé par rapport à une modélisation transactionnelle d'un système. Ici, nous souhaitons aborder la modélisation des parties supérieures de l'arbre d'horloge, celles qui font du sens dans une vue système.

Pour atteindre notre objectif, nous avons développé un script TCL qui réalise une structuration de l'architecture selon la stratégie *ClkARCH* en ajoutant à une description IP-XACT préalablement définie les informations nécessaires pour une décomposition en *Clock Domains*. Une première approche de description d'un *clock intent* en IP-XACT serait d'utiliser les principes retenus ci-dessus à propos des *Power Domains* et de les transposer aux *Clock Domains* :

- *Voltage Domain* ou valeur de *Voltage* correspondrait à l'horloge source et donc une DPLL.
- Un *Power Domain* correspondrait à un *Clock Domain*.

En revanche, une première difficulté apparaît à savoir qu'un *Clock Domain* peut avoir deux horloges sources, une horloge de référence et une horloge de dérivation (*Bypass*). La hiérarchie dans les *Clock Domains* n'existe pas du fait de la propriété de synchronisation d'horloge qu'il y aurait entre l'horloge d'un domaine parent et celle d'un domaine fils. Enfin, la notion de *Clock Domain* ne s'appuie actuellement sur aucun standard, aussi intégrer dans *Platform Assembly* l'approche considérée pour les *Power Domains* aux *Clock Domains* n'apparaît pas actuellement pertinent. Par conséquent, nous avons opté pour une solution moins adhérente à l'outil en utilisant exclusivement les *Vendor Extensions* d'IP-XACT et sans affecter la base de données interne de l'outil.

Le principe est d'associer directement à chaque composant du *Design* des *Vendor Extensions* qui renseignent le nom du *Clock Domain* et le ou les noms des sources d'horloge associées.

```
aiprtl:: AddAttribute -comp "/G711_Encoder_inst_0" -attribute clock: domain -value "ClkD_AUDIO"
aiprtl:: AddAttribute -comp "/G711_Encoder_inst_0" -attribute clock: source -value "Clk_Audio_R" -type "Reference"
aiprtl:: AddAttribute -comp "/G711_Encoder_inst_0" -attribute clock: source -value "Clk_Audio_B" -type "Bypass"
aiprtl:: AddAttribute -comp "/G711_Encoder_inst_0" -attribute clock: dpll -value "DPLL_AUDIO"
```

Figure IV.10 – Script TCL pour la structuration de la plateforme selon *ClkARCH*

```
<mdsAttr:group mdsAttr:groupName="clock">
    <mdsAttr:attribute mdsAttr:name="domain">
        <mdsAttr:value mdsAttr:prompt="Root">ClkD_AUDIO</mdsAttr:value>
    </mdsAttr:attribute>
    <mdsAttr:attribute mdsAttr:name="source">
        <mdsAttr:value mdsAttr:prompt="Root" mdsAttr:type="reference">Clk_Audio_R</mdsAttr:value>
    </mdsAttr:attribute>
    <mdsAttr:attribute mdsAttr:name="source">
        <mdsAttr:value mdsAttr:prompt="Root" mdsAttr:type="bypass">Clk_Audio_B</mdsAttr:value>
    </mdsAttr:attribute>
    <mdsAttr:attribute mdsAttr:name="dpll">
        <mdsAttr:value mdsAttr:prompt="Root">DPLL_AUDIO</mdsAttr:value>
    </mdsAttr:attribute>
</mdsAttr:group>
```

Figure IV.11 – IP-XACT créé pour décrire une structuration selon *ClkARCH*

L'extrait du script TCL dans la Figure IV.10 montre que nous associons une instance de composant à un *Clock Domain*. Ensuite, nous spécifions les horloges sources externes de type *Reference* et *Bypass* associées au *Clock Domain*. Finalement, nous définissons une DPLL. L'exécution de ce script permet de créer la description correspondante en IP-XACT. La description

de structuration en *ClkARCH* vient s'insérer dans le *Design* sous forme de *Vendor Extension* (Figure IV.11).

En conséquence, nous obtenons une représentation clairement définie qui permet de structurer une architecture construite par assemblage d'IP selon un *power intent* (*PwARCH*) et un *clock intent* (*ClkARCH*). Cette structuration est simulable si nous créons le code SystemC-TLM/C++ associé. Dans le paragraphe suivant, nous décrivons les étapes de génération de code SystemC-TLM/C++.

4 Génération de la description en C++ d'une stratégie de gestion de puissance

Structurer manuellement des architectures selon des stratégies de gestion de puissance nécessite un effort de modélisation et peut conduire à des erreurs de conception peu triviales à identifier parce qu'elles peuvent résulter des comportements incohérents entre ceux induits par les décisions issues de la gestion de puissance et ceux relatifs aux états fonctionnels produits par l'exécution du modèle SystemC-TLM. Ce problème de vérification est déjà identifié comme un thème de recherche même au niveau TLM [MM13]. Ainsi, pour réduire le nombre d'erreurs, le but est d'automatiser la génération de code relative à la description d'une structure de gestion de puissance dans une architecture.

Un générateur d'une description structurelle en SystemC-TLM d'un design décrit en IP-XACT sous *Platform Assembly* a été développé par Magillem et rendu accessible dans cet environnement. Ce générateur produit une description structurelle SystemC-TLM de la partie fonctionnelle. Comme notre approche s'appuie sur une séparation des préoccupations fonctionnelle et *power/clock* (Figure IV.12), le *power/clock intent* est majoritairement supporté par une librairie C++ et non SystemC-TLM. Uniquement le PMU qui est le composant effectuant l'interface de contrôle entre le modèle fonctionnel logiciel/matériel et le modèle de gestion de puissance consommée est décrit en SystemC-TLM. Par conséquent, nous avons introduit un nouveau générateur via *TGI Workshop* pour obtenir une description suivant un code en C++ du *power/clock intent*.

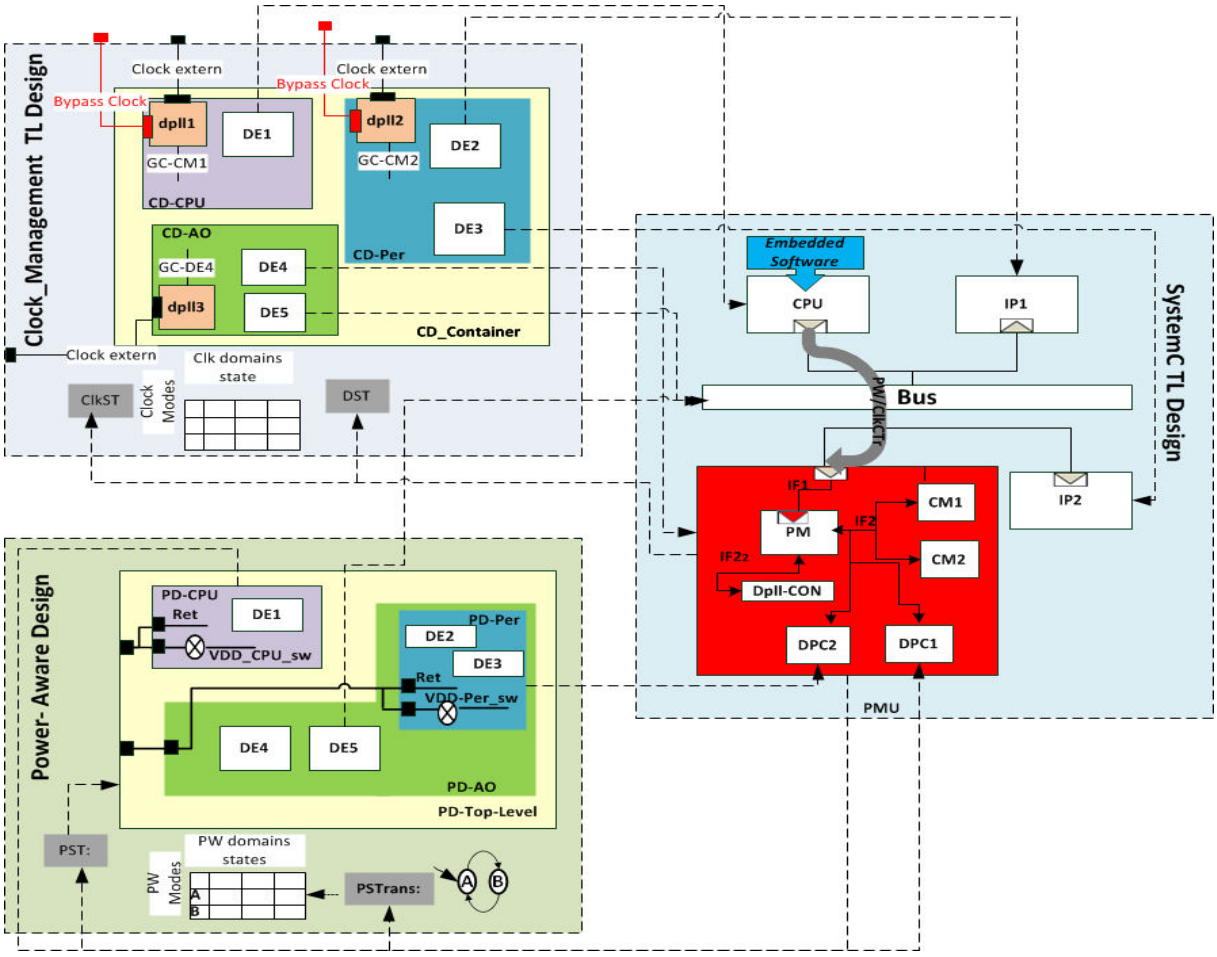


Figure IV.12 – Architecture SystemC-TLM augmentée par un *clock/power intent*

4.1 Extraction des informations relatives au *power intent* et au *clock intent*

Ayant toutes les informations décrivant à la fois l'aspect fonctionnel (réduit à la structure) et l'aspect gestion de consommation de puissance dans le modèle IP-XACT, nous avons conçu un générateur de code C++ qui extrait les informations utiles pour une génération de la description de la structuration de l'architecture selon la stratégie de gestion de puissance renseignée dans la description IP-XACT. Ce générateur extrait les informations de la base de données pour automatiser la création de la partie *Power_Main* (Figure IV.13). Ce code instancie les composants ad-hoc depuis les librairies C++ *ClkARCH* et *PwARCH* pour décrire l'approche de gestion de puissance au niveau transactionnel. Nous avons utilisé les fonctions implémentées par l'API TGI qui nous permettent d'extraire les différentes données renseignées dans les

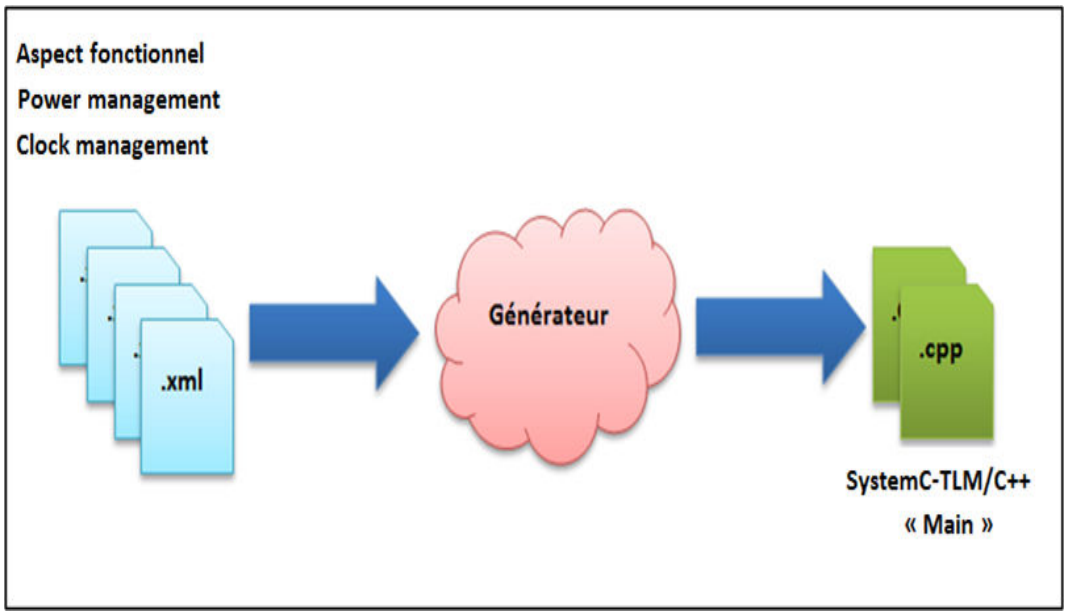


Figure IV.13 – Générateur de code *Power_Main* SystemC-TLM/C++

composants et le *Design* au moment de la description de la plateforme en IP-XACT. Cette API est intégrée dans l'environnement Magillem. Ses fonctionnalités sont accessibles à travers les éléments de type *generator* qu'il est nécessaire de créer dans le module *TGI Workshop* de Magillem.

Différents langages de programmation (Java, TCL, Python et Ruby) peuvent être utilisés pour développer un générateur de code. Nous avons développé notre générateur de code en Java. En effet, les scripts Java sont à privilégier pour les *generators* complexes. Ils sont compatibles avec différentes plateformes. De plus, un *Java Development Toolkit* basé sur *Eclipse* est intégré à l'environnement Magillem.

Les informations relatives au *power intent* et au *clock intent* sont décrites dans les éléments *Vendor Extensions* qui sont entièrement personnalisables par l'utilisateur du standard. Cependant, les fonctions définies par la TGI ne permettent pas l'extraction de ces données autrement qu'en texte brut, c'est-à-dire sous forme d'un texte XML. Il fallait donc implémenter nos propres fonctions pour extraire et traiter les balises et les attributs associés pour la génération de code. Nous avons utilisé le standard *Document Object Mode* (DOM) [DOM] pour traiter les *Vendor Extensions* afin d'y extraire les informations nécessaires pour la génération de code.

4.2 Génération de code C++

En traitant les métadonnées de la base de données IP-XACT, nous avons extrait les informations utiles et développé un algorithme pour traiter ces informations, établir les relations entre elles et générer la description en C++ de la structure du *clock intent* et du *power intent*.

Dans un *clock/power intent*, chaque bloc IP de l'architecture fonctionnelle appartient à la fois à un *Clock Domain* et à un *Power Domain*. Comme le *Design_Element* représente le pointeur vers ce bloc IP, nous l'associons à un *Clock Domain* et à un *Power Domain*. La Figure IV.14 montre une partie de l'instanciation des *Clock Domains*, des *Power Domains* et des *Design_Elements* de l'architecture de la plateforme Audio suivant la décomposition de la Figure IV.9 et la Figure III.25.

```
/*****************create Power Domains*****************************/
/******************************************************/

Power_Domain* PD_AO= new Power_Domain(PD_Container, "PD_AO" , "Scope1" );
Power_Domain* PD_ENC= new Power_Domain(PD_AO, "PD_ENC" , "Scope1" );
Power_Domain* PD_DEC= new Power_Domain(PD_AO, "PD_DEC" , "Scope1" );
.......
/*****************create Clock Domains*****************************/
/******************************************************/

Clock_Domain* CD_NATIVE= new Clock_Domain(CD_Container, "CD_NATIVE");
Clock_Domain* CD_AUDIO= new Clock_Domain(CD_Container, "CD_AUDIO");
.......
/*****************create Design Elements*****************************/
/******************************************************/

//Design_elem(Power_Domain* pDomain_name, Clock_Domain* pDomain_name,
//sc_core::sc_object& p_obj, float activity, float cap, float leakage);
Design_elem DE_G711_Encoder= new Design_elem (PD_ENC,CD_AUDIO,G711_Encoder,1,100,0.7);
Design_elem DE_G711_Decoder= new Design_elem (PD_DEC,CD_AUDIO,G711_Decoder,1,100,1.2);
Design_elem DE_CPU= new Design_elem (PD_AO,CD_NATIVE,CPU,1,120,1.8);
```

Figure IV.14 – Description de la décomposition en *Power Domains* et *Clock Domains*

Pour fournir aux composants de la plateforme décomposée en *Power* et *Clock Domains* les valeurs de tension d'alimentation et de fréquence d'horloge, il est nécessaire de définir les *Supply Nets* et les DPLL. Or, comme nous l'avons déjà indiqué, chaque DPLL peut avoir besoin de deux sources d'horloges externes : une de référence et une autre de dérivation (*Bypass*). À travers notre générateur nous instancions des DPLL en les associant aux *Clock Domains* correspondants et nous créons également les horloges d'entrée des DPLL. Ceci est illustré sur la Figure IV.15.

Contrairement aux éléments déjà cités, les *Clock Managers* ne sont pas représentés dans

```
/*****************create Supply Net ****************************/
/*****************************************************/
Supply_Net* VDDSOC=new Supply_Net("VDDSOC", "power_net",PD_AO);
Supply_Net* VDDENC=new Supply_Net("VDDENC", "switched",PD_ENC);
Supply_Net* VDDDEC=new Supply_Net("VDDDEC", "switched",PD_DEC);


/************** Create DPLL and  its input clocks *****/
/*****************************************************/
DPLL* DPLL_Audio =new DPLL(CD_AUDIO,"DPLL_Audio");
Clock_Externe* Clk_Audio_R=new Clock_Externe("Clk_Audio_R","reference",DPLL_Audio,200);
Clock_Externe* Clk_Audio_B=new Clock_Externe("Clk_Audio_B","Bypass",DPLL_Audio,60);
```

Figure IV.15 – Instanciation des sources d'alimentation

la description IP-XACT. En effet, selon la structure de *ClkARCH*, un *Clock Manager* est un composant à l'intérieur du PMU. Son instance est créée au moment de l'instanciation du PMU.

Un *Clock Manager* est alimenté par une horloge d'entrée qui est générée par une DPLL associée au même *Clock Domain* que lui. Un *Clock Manager* produit et gère les horloges des *Design_Elements* inclus dans un même *Clock Domain*. Pour automatiser cette distribution d'horloge, nous avons générer pour chaque DPLL et *Design_Element* une déclaration d'une instance d'horloge *Generated_Clock* (Figure IV.16). Ces instances sont affectés au *Clock Manager* automatiquement au moment de sa création.

```
/*******************create Generated Clock*************************/
/*****************************************************************/
// Generated Clock from DPLL to Clock Manager
Generated_Clock* DPLL_CM0=new Generated_Clock("DPLL_CM0","Output",DPLL_Native);
Generated_Clock* DPLL_CM1=new Generated_Clock("DPLL_CM1","Output",DPLL_Audio);
.....
// Generated Clocks from Clock Manager to Design Elements
Generated_Clock* CM1_DE_CPU=new Generated_Clock("DPLL_CM1","Input",DE_CPU);
Generated_Clock* CM1_DE_BUS=new Generated_Clock("DPLL_CM1","Input",DE_BUS);
.....
```

Figure IV.16 – Description de la structure des horloges de type *Generated Clock*

Par ailleurs, nous avons choisi de ne pas ajouter les *power switchs* à la description IP-XACT de l'architecture. En effet, à partir des informations déjà introduites dans la représentation IP-XACT, nous pouvons déduire la génération de code nécessaire à leur instanciation. Le code

généré relatif au *power switchs* est basé sur la règle définie dans *PwARCH* : pour chaque *Power Domain* dont le *Supply Net* est de type *switched*, nous associons un *power switch*. Un exemple de l'instanciation des *power switchs* apparait dans la Figure IV.17 avec l'instanciation des tensions d'alimentation d'entrée et de sortie associées à chaque *power switch*.

```
/*****************create Power Switch ****************************/
/*****************************************************************/

Power_Switch* PSW1 = new Power_Switch(PD_ENC,"SW1");
 (SW1->_input_supply_nets).push_back(VDDSoC);
  SW1->SetOutput_supply_net(VDDENC);

Power_Switch* PSW2 = new Power_Switch(PD_DEC,"SW2");
 (SW2->_input_supply_nets).push_back(VDDSoC);
  SW2->SetOutput_supply_net(VDDDEC);
```

Figure IV.17 – Instances des *power switchs*

Dans cette section, nous avons décrit les différentes méthodes utilisées et proposées pour effectuer la génération d'une description de structuration d'une architecture selon un *power intent* et un *clock intent*. Le code obtenu n'est pas encore complet pour effectuer une simulation globale du système. En effet, il manque l'instanciation du PMU dans le modèle SystemC-TLM, composé du *Power Manager*, des *Clock Managers*, des *Domain Power Controller* et du *DPLL_ Controller*. Une approche basée sur l'ingénierie des modèles a été utilisée dans [Ame15] pour produire le code SystemC-TLM du PMU suivant l'approche de modélisation définie dans cette étude. Cette génération s'appuie sur le fait qu'un *Clock Manager* est associé à chaque *Clock Domain*, qu'un *Domain Power Controller* est associé à chaque *Power Domain*, que l'ensemble des DPLL est géré par un seul *DPLL_Controller*, le tout sous le contrôle du *Power Manager* connecté au bus système.

En définitive, la seule partie de code manquante concerne le comportement du PM et la définition des contenus des tables PST, ClkST et DST qui sont dépendantes de la stratégie de gestion de puissance.

5 Application du *clock/power intent* sur la plateforme Audio

Pour illustrer cette fonctionnalité de génération de code structurel lié aux spécifications du *power intent* et du *clock intent*, nous reprenons l'exemple de la plateforme Audio décrite dans le

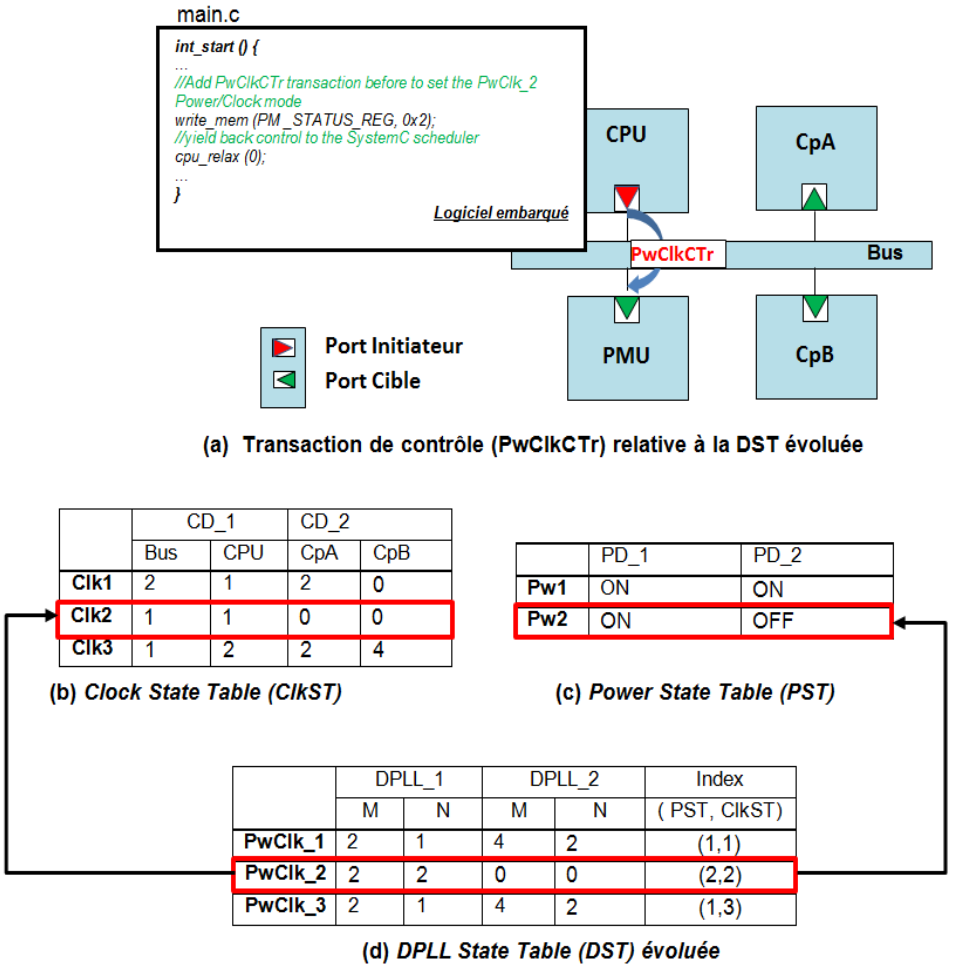
chapitre III. L'objectif est ici de simuler la plateforme complète structurée suivant une stratégie de gestion des *Clock Domains* et *Power Domains*.

La description structurelle obtenue par la génération du code est basée d'une part sur les éléments de la librairie *ClkARCH* qui définit un *clock intent* et d'autre part la librairie *PwARCH* qui décrit un *power intent*. Dans le but de gérer à la fois le *power/clock intent*, nous avons fusionné les deux libraires *PwARCH* et *ClkARCH* en une seule librairie appelée *PwClkARCH*. Nous avons utilisé cette librairie dans une stratégie de gestion de puissance basée à la fois sur la décomposition en *Clock Domains* et la décomposition en *Power Domains*. Nous décrivons dans la suite cette stratégie.

5.1 Stratégie de gestion des *Clock/Power Domains*

À ce niveau, nous nous intéressons à une combinaison de deux stratégies. D'une part, nous avons décrit dans le chapitre III la stratégie de gestion de puissance basée sur la décomposition en *Clock Domains*. Nous avons associé à chaque *Clock Domain* un *Clock Manager* qui gère les différents modes de puissance du système définis dans la *Clock State Table* (ClkST). De plus, un *DPLL_Controller* est mis en place pour gérer les DPLL à travers la *DPLL State Table* (DST). D'autre part, dans la section 3.3, nous avons décrit brièvement la stratégie de gestion de puissance basée sur les *Power Domains*. Un *Domain Power Controller* (DPC) est associé à chaque *Power Domain* afin de gérer les modes de puissance définis dans la *Power State Table* (PST). Ainsi, pour gérer les *Power Domains* et les *Clock Domains*, nous avons considéré la même approche en se basant sur des tables à savoir la ClkST et la PST et une évolution de la DST. Pour ce faire, par rapport à la structuration proposée du PMU dans le chapitre III (Figure III.22) qui contient les *Clock Managers* et le *DPLL_Controller*, il a été nécessaire d'introduire également dans le PMU les *Domain Power Controllers* dans le même *Power Management Unit* comme l'illustre la Figure IV.12. Tous ces composants sont commandés par le *Power Manager*.

Le PMU est modélisé comme un bloc fonctionnel SystemC-TLM qui communique à travers le bus par des transactions TLM avec les autres blocs IP de la plateforme comme le montre la Figure IV.18(a). Déclencher le changement d'un mode de puissance correspond à une demande de gestion de puissance représentée par une transaction TLM issue d'un initiateur dans le prototype virtuel (par exemple le CPU). Ce type de demande est notée *PwClkCTr* (voir Figure IV.18). En fait, cette demande correspond à une référence à une ligne de la DST évoluée. La DST initiale définit les facteurs de division et de multiplication associes aux différents états des DPLL. Pour mettre en place une stratégie de *power/clock intent*, nous avons ajouté à la DST une colonne qui contient deux références : une vers une ligne de la PST et une autre vers



	CD_1		CD_2	
	Bus	CPU	CpA	CpB
Clk1	2	1	2	0
Clk2	1	1	0	0
Clk3	1	2	2	4

	PD_1	PD_2
Pw1	ON	ON
Pw2	ON	OFF

	DPLL_1		DPLL_2		Index
	M	N	M	N	(PST, ClkST)
PwClk_1	2	1	4	2	(1,1)
PwClk_2	2	2	0	0	(2,2)
PwClk_3	2	1	4	2	(1,3)

Figure IV.18 – Exemple d'un scénario appliquant la stratégie basée sur les *Power Domains et Clock Domains*

une ligne de la ClkST (voir Figure IV.18). Ainsi, la référence à une ligne de cette DST évoluée correspond à un OPP pour lequel nous définissons les états des *Clock Domains* et des *Power Domains* mettant en œuvre cet OPP.

Un changement d'OPP peut conduire à des pénalités en temps (et en énergie) qui dépendent du changement d'état de tension/fréquence demandé vis-à-vis de l'état courant. Pour tenir compte des pénalités induites et de la procédure classiquement utilisée dans les systèmes, nous opérons comme suit.

A la réception d'une demande de changement de mode de puissance, deux scénarios peuvent avoir lieu comme le montre la Figure IV.19.

– Si la fréquence demandée est supérieure à la fréquence courante, le PM commence par

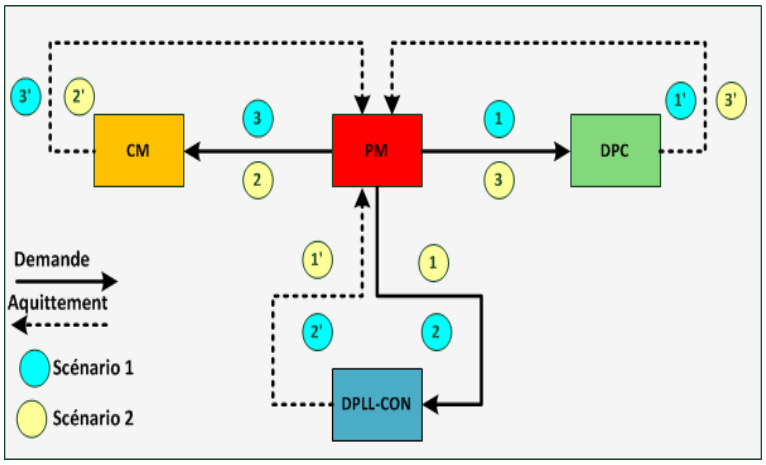


Figure IV.19 – Scénario de la gestion du *clock/power intent*

mettre à jour les valeurs de la tension selon la ligne demandée de la PST à travers les DPC. A la réception d'un acquittement, le PM envoie une demande au *DPLL_Controller* pour changer les états des DPLL. Ceci commence par la sélection de la fréquence de dérivation (*Bypass*) des DPLL concernés et par le changement des facteurs de division/multiplication suivant la ligne de la DST référencée. Enfin, le PM commande les CM pour modifier les fréquences de leurs *Clock Domains* suivant la ligne correspondante de la ClkST.

– Si la fréquence demandée est inférieure à la fréquence courante, le PM commence par sélectionner l'horloge de dérivation des DPLL concernées puis par changer les états des DPLL à travers le *DPLL_Controller*. Ensuite, il met à jour les valeurs de la fréquence selon la ligne demandée de la ClkST en interrogeant les CM après la réception d'un acquittement du *DPLL_Controller*. Enfin, le PM demande aux DPC de modifier les valeurs de tension de leurs *Power Domains* suivant la ligne correspondante de la PST.

Pendant le changement de fréquence via les DPLL, les *Design_Elements* voient leur fréquence égale à la fréquence de dérivation, c'est-à-dire que pendant cet intervalle de temps, les vitesses de traitement des blocs IP vont être ralenties.

La stratégie de la gestion du *power/clock intent* respecte le flot de notre méthodologie décrite dans le chapitre III. Par conséquent, il est nécessaire de définir d'autres contrats de vérification qui s'adresse plutôt à la relation entre le *Power Domains* et les *Clock Domains*. Ce sont des contrats de type 2 :

– La fréquence d'un *Design_Element* ne peut pas changer si ce dernier est inclus dans un *Power Domain* inactif (OFF).
– Un *Clock Domain* ne peut être actif que lorsque sa DPLL est active. Donc, elle est incluse

dans un *Power Domain* actif.

Des contrats plus spécifiques pourraient être considérés vérifiant par exemple des niveaux de tension minimum par rapport à une valeur de fréquence afin de vérifier la validité d'un OPP. Cependant ce type de contrat fait intervenir des aspects technologiques qui sont peu génériques actuellement. Si la version de UPF3.0 à venir contient des informations par bloc IP permettant de mettre en place ce type de vérification, de tels contrats pourront être intégrés aisément.

5.2 Résultats de la simulation de la plateforme Audio augmentée par la combinaison *clock/power intent*

L'objectif ici est de combiner le code généré décrivant la structure de la plateforme Audio selon la décomposition en *Clock Domains* (Figure III.25) et selon la décomposition en *Power Domains* (Figure IV.9) avec le *Power Management Unit* qui implémente la stratégie de gestion de puissance relative au scénario Audio *Recorder* décrit dans la section 4.1. L'application de cette stratégie de gestion du *power/clock intent* avec la plateforme Audio conduit à la consommation de la puissance statique illustrée sur la Figure IV.20. Le contenu de la PST pour cet

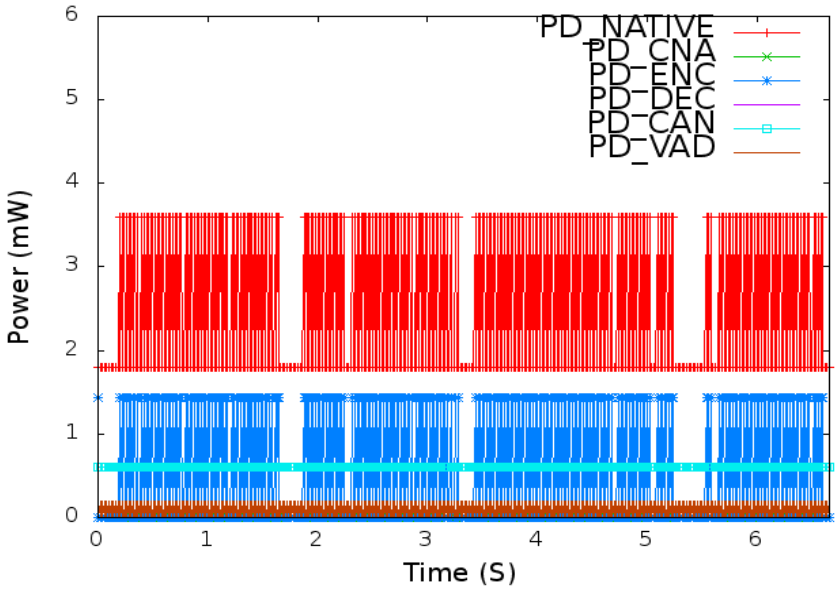


Figure IV.20 – Puissance statique consommée pendant l'exécution du scénario Audio *Recorder* sur la plateforme Audio augmentée par un *power/clock intent*

exemple est donné sur la Figure IV.21. Cette PST fait intervenir des états différents ayant une tension est non nulle pour un *Power Domain*, en particulier *PD_NATIVE*. Ceci se traduit par

une valeur correspondante du *supply net* qui alimente le *Power Domain* autorisant des valeurs de fréquence différentes (et donc différents OPP) en fonction de l'état considéré de ce Power Domain. Le contenu de la ClkST est donné dans la Figure III.26

	VDD_NATIVE	VDD_ENC	VDD_DEC	VDD_VAD	VDD_CAN	VDD_CNA
Read_from_file	ON-Low	OFF	OFF	ON	ON	OFF
Voice Detection	ON-Low	OFF	OFF	OFF	ON	OFF
Encoding	ON-High	ON	OFF	OFF	ON	OFF
Write results in memory	ON-High	OFF	OFF	OFF	ON	ON

Figure IV.21 – *Power State Table* du scénario Audio *Recorder*

La consommation statique varie d'un *Power Domain* à un autre puisque avec notre stratégie, chaque *Power Domain* peut être contrôlé individuellement. La puissance statique du *Power Domain PD_NATIVE* varie entre un minimum non nul et une valeur maximum, cela correspond aux deux états de la PST pour ce *Power Domain* avec les valeurs associées du *supply net*. Pour connaître la puissance totale dissipée par l'architecture, nous associons à la puissance dynamique représentée sur la Figure III.27 la puissance statique de la Figure IV.20. Par ailleurs, durant la simulation nous avons vérifié que les contrats spécifiés ci-dessus sont bien vérifiés, à savoir qu'une fréquence active (issue d'une DPLL ou appliquée à un DE) correspond bien à un *Power Domain* à l'état ON. Enfin, pour évaluer l'impact sur le temps de simulation de l'introduction du modèle du *power/clock intent* dans l'architecture fonctionnelle, nous avons mesuré le temps de simulation du modèle instrumenté avec le *power/clock intent* par rapport au modèle fonctionnel pur. L'accroissement du temps de simulation sur une séquence d'encodage de 7 secondes de parole est de 3.5% ce qui reste faible compte tenu de l'apport de la méthode proposée permettant l'évaluation et la vérification d'une stratégie complète de gestion de puissance statique et dynamique.

6 Conclusion

Dans ce chapitre nous avons souhaité montrer que l'approche proposée pour décrire et modéliser un *clock intent*, associé à un *power intent*, peut s'intégrer dans une démarche outillée basée sur un standard du domaine (IP-XACT). La possibilité de considérer des *Vendor Extensions* du standard a été utilisée pour décrire ces aspects orientés gestion de puissance puisque les versions actuelles de ces standards ne supportent pas ce type de description. Néanmoins, cette étude

montre que la quantité d'information à ajouter à une description fonctionnelle pour décrire une décomposition en *Clock Domains* et *Power Domains* est relativement restreinte pour obtenir une vue au niveau TLM d'un *clock/power intent*. Avec la possibilité offerte par IP-XACT de proposer différentes vues d'un système (vue RTL et vue TLM), IP-XACT apparaît comme une approche intéressante de description structurelle facilitant alors le passage d'une vue TLM d'un *clock/power intent* à une vue RTL qui induit alors d'autres éléments plus raffinés. La première approche développée par Magillem pour instancier automatiquement les cellules de type *level shifters* ou *isolation cells* illustre bien que ce passage fait du sens.

Pour récapituler, dans ce chapitre et le chapitre précédent, nous avons proposé une approche de gestion de puissance pour la réduction de la consommation basée essentiellement sur la séparation entre les préoccupations fonctionnelles et non fonctionnelles. Cette approche permet la modélisation et la vérification des stratégies de de gestion de puissance et d'horloge (*clock intent* et *power intent*) en utilisant la librairie *PwClkARCH*. Ces stratégies considèrent une vue centralisée des états globaux définis dans la table ClkST (identifier les différents fréquences des composants) et la table PST (identifier les voltages des composants) relatives au scénario exécuté sur une architecture donnée. Pour gérer ces tables, une stratégie statique est implémentée dans le *Power Management*. Pour rendre cette stratégie dynamique et indépendante de la connaissance de l'application exécutée, nous proposons dans le chapitre suivant une nouvelle approche de la gestion de puissance. Celle-ci est basée sur l'analyse des états fonctionnels pour identifier dynamiquement l'état de puissance de chaque composant et à partir de ces états fonctionnels nous explicitons les relations entre états fonctionnels et les contrôles des PD et CD.

Chapitre V

Stratégie de gestion de puissance pour les systèmes sur puce à faible consommation basée sur l'analyse des états fonctionnels

1 Introduction

Le défi des concepteurs des systèmes sur puce aujourd'hui est non seulement de livrer un produit fonctionnel mais aussi de satisfaire un ensemble de contraintes non fonctionnelles dont la gestion de puissance fait partie. En effet, la consommation d'énergie des SoC augmente de plus en plus parallèlement au nombre croissant de blocs IP intégrés pour répondre aux exigences de performances des applications embarquées. La gestion de puissance de ces systèmes devient également difficile à optimiser avec les nombreux contrôles et contraintes imposés par les différentes couches du système (application / logiciel / matériel) tels que la gestion de QoS des applications, le système d'exploitation et les pilotes de périphériques. L'intégration d'une stratégie de gestion de puissance efficace traversant toutes ces couches a tendance à ne pas être générique du fait de l'absence d'une définition d'interfaces normalisées entre ces couches. Ceci est par exemple le cas des systèmes basés sur Linux [KVG+12], où la décision d'éteindre les composants autres que les cœurs du CPU, et gérés par un pilote Linux, est prise en se basant sur une analyse locale de l'état fonctionnel du composant ce qui limite les moyens de réaliser des économies d'énergie efficaces. En effet, l'analyse de l'état local d'un composant conduit à ignorer d'éventuelles dépendances d'états qui peuvent exister entre composants en fonction du scénario applicatif à exécuter. Ces dépendances peuvent par exemple conduire à ce qu'un composant B (un DMA par exemple) dans un état inactif devienne à courts termes actif du fait qu'un autre composant A dans l'architecture (un sous-système audio par exemple) est actuellement dans un état actif. Ce type de dépendance milite pour que l'analyse des états

soit étendue au-delà d'une analyse locale par composant pour limiter les mauvaises décisions de gestion de puissance. Traditionnellement, la gestion de puissance se base sur les techniques de réduction d'énergie qui sont le *power gating* (2.5.3.1), le *clock gating* (2.5.3.2) et DVFS (2.5.3.3). Les techniques du *power gating* et DVFS permettent des économies d'énergie seulement si les conditions de temps de type *break-even-time* T_{be} (II.3) sont remplies. La valeur de T_{be} est la borne inférieure du temps d'inactivité du bloc IP pour compenser le coût de l'énergie impliqué par le changement de mode de puissance (*power gating*/DVFS) appliqué au bloc IP. Elle est directement dépendante des caractéristiques physiques de l'architecture matérielle. Rappelons qu'en appliquant la technique de *power gating*, les puissances statique et dynamique des blocs IP deviennent nulles. La technique de DVFS réduit à la fois la consommation statique et dynamique tandis que la technique du *clock gating* coupe seulement la puissance dynamique.

Dans ce chapitre, nous proposons une stratégie de gestion de puissance basée uniquement sur l'analyse de l'historique des états fonctionnels globaux des blocs IP ou des *Power Domains*. L'observation de ces états fonctionnels fournit une information condensée sur l'ensemble des éléments qui interviennent dans le processus d'exécution d'une application sur l'architecture. Dans l'approche proposée, les blocs IP peuvent être contrôlés individuellement par l'unité de gestion de puissance (PMU) ou regroupés en *Power Domains* et/ou en *Clock Domains*. Une gestion en considérant les *Power Domains* ou les *Clock Domains* permet de limiter la taille du problème lorsque le nombre de blocs IP est important dans l'architecture (plusieurs dizaines par exemple). Cette stratégie ne dépend pas du système d'exploitation ou de l'application tandis qu'elle peut être contrôlée facilement par ces couches logicielles.

2 Stratégie de la gestion de puissance basée sur l'analyse des états fonctionnels

De nombreux travaux ont été publiés visant à optimiser la consommation d'énergie ou de puissance dans des systèmes sur puce en tenant compte de contraintes de performances et des pénalités induites par les changements de mode de puissance des blocs IP du système. Dans ces travaux, la gestion de puissance est abordée soit par une analyse hors-ligne, soit par des approches s'exécutant en ligne en observant les comportements, soit de manière mixte. Dans les approches hors ligne, les caractéristiques de l'application et de l'architecture sont considérées pour élaborer une stratégie de gestion de puissance. L'architecture et l'application sont alors représentées par des modèles abstraits et la stratégie de gestion de puissance est alors obtenue par une solution d'un problème d'optimisation intégrant également des contraintes

d'affectation de tâches et d'ordonnancement. Nous citons l'exemple de l'approche proposée dans [Ca13] qui utilise une heuristique pour déterminer une solution alors qu'une formulation de type programme linéaire en nombres entiers est proposée dans [CHK14]. Cette dernière considère un graphe de flots de données acyclique pour décrire l'application et une architecture supportant DPM et DVFS. Dans ce type d'approche hors ligne, la stratégie de gestion de puissance est dédiée à l'application considérée avec un comportement dans le pire cas (pire cas des temps d'exécution des tâches). Pour des systèmes temps réels spécifiques embarqués, ce type d'approche peut convenir mais lorsque la dynamique du système est importante il devient peu efficace.

Dans le cas d'approches de gestion de puissance opérant en ligne, les décisions sont prises sur la base de l'observation des états des unités du système et d'informations issues du niveau applicatif. Par exemple, dans [MPV+13a], une approche par théorie du contrôle est considérée mais son efficacité dépend de la connaissance a priori sur l'orientation éventuelle de type *QoS* des tâches à exécuter et des taux d'utilisation du CPU de chaque tâche. Dans [MSKD10], une approche basée sur la théorie du contrôle est aussi utilisée où une gestion de puissance à deux niveaux est définie. L'objectif est de maximiser le débit des instructions exécutées suivant un budget puissance maximum. Une gestion de puissance au niveau système d'exploitation est développée dans [Ba06] mais cette stratégie nécessite une connaissance a priori des composants du système qui sont utilisés par chaque tâche de l'application. La technique détaillée dans [JSHP11] est une approche mixte hors-ligne/en-ligne pour des architectures MPSoC qui exploite un pipeline d'exécution pour des applications flots de données. Les nombres minimum de périodes d'inactivité des composants sont évalués hors-ligne pour l'application considérée ce qui permet ensuite de déduire une charge de calcul permettant alors de gérer en ligne les états de puissance des composants. Ce survol des travaux dans le domaine montre que ces approches impliquent des connaissances a priori en particulier sur les applications à exécuter ce qui limite ainsi leur utilisation dans le cas de systèmes devant supporter une variété d'applications.

Nous détaillons dans la suite la stratégie de gestion de puissance proposée qui est structurée en un processus de construction d'un graphe de l'historique des états fonctionnels, d'un processus de décision de gestion de puissance et d'une procédure de vieillissement du graphe de l'historique. Nous commençons par définir un ensemble de termes et de notations.

Nous notons par le terme tâche une activité qui peut être associée à un code logiciel ou à une fonctionnalité matérielle exécutée sur un bloc IP (par exemple, un CPU, un accélérateur matériel, etc). Une architecture est composée de n blocs IP ou n *Power Domains* notés x_0, x_1, ..., x_{n-1}. Nous notons que t_{be,x_j} définit le temps de type *break-even-time* du bloc IP x_j. Si un bloc IP a plus d'un mode de puissance (par exemple, *Idle*, Actif, etc), un *break-even-time*

constant est associé à chaque mode de puissance.

La consommation d'énergie est produite par les blocs IP sur lesquels des tâches de la ou des applications sont exécutées. L'état fonctionnel (actif ou inactif) de chaque bloc IP est la conséquence de la combinaison d'actions de création, suppression, *mapping*, ordonnancement et migration effectuées par le système d'exploitation et/ou par le *middleware* (le cas échéant) sur les tâches applicatives. Dans ce contexte, une stratégie de gestion de puissance basée sur des informations et des métriques venant directement de l'application logicielle par une caractérisation hors ligne de l'application semble être très difficile à mettre en œuvre. En effet, ces informations devraient dans ce cas être actualisées dynamiquement suivant les décisions d'exécution prises par le système (ordonnancement, migration, ...) pour en déduire justement ces mêmes décisions en les couplant avec celles de gestion de puissance. Cependant, observer les états des blocs IP dans l'architecture est équivalent à observer le résultat de la composition des fonctions de *mapping*/migration et d'ordonnancement appliquées à l'ensemble des tâches, quelle que soit l'application considérée. Nous désignons par $s_j(t)$ l'état actif ou inactif du bloc IP x_j.

Nous considérons $x_j = M(t, \tau_k)$ la fonction de *mapping*/migration qui associe à un instant t le bloc IP x_j avec la tâche τ_k. L'état $s_j(t)$ du bloc x_j à l'instant t est défini par :

$$s_j(t) = \bigcup S(M(t, \tau_k), t) / \ \forall \tau_k \in T \ et \ M(t, \tau_k) = x_j; \tag{V.1}$$

avec $S(x_j, t)$ est une fonction booléenne qui présente la politique de l'ordonnancement et T est l'ensemble des tâches de l'application.

La fonction $S(x_j, t)$ est vraie si la tâche est exécutée sur le bloc IP x_j à l'instant t sinon la fonction est fausse. Nous désignons par le vecteur S(t) = ($s_0(t)$, $s_1(t)$,..., $s_{n-1}(t)$) l'état global de l'architecture à l'instant t.

À l'instant courant t_c, l'analyse de l'historique des états successifs S(t), avec $t < t_c$, fournit une information pertinente pour prédire les prochains états actifs/inactifs des blocs IP dans l'architecture. Cette analyse d'état global S(t) permet d'étudier la corrélation entre les états individuels des blocs IP qui servent à prendre une décision efficace de prédiction. Notre stratégie de gestion de puissance permet d'appliquer essentiellement les modes de puissance : *clock gating* et *power gating*, mais elle ne considère pas la technique de DVFS. En effet, cette technique dépend principalement des exigences de performance de l'application et peut être traitée plus efficacement au niveau de l'application ou du système d'exploitation (par exemple, la fonction CPUfreq () sous Linux). Cependant, la stratégie de *power gating* doit être robuste aux décisions de gestion de DVFS (la fréquence d'horloge peut être diminuée ou augmentée par *Clock Domain*)

afin de maintenir des gains en puissance ou en énergie efficaces même en présence de pénalité en temps dues au changement de mode de puissance et tout en n'affectant pas les performances visées en fonction de la fréquence d'horloge fixée.

2.1 Construction d'un graphe de l'historique des états globaux (*History Graph* - HG)

Un graphe décrivant l'historique des états liste l'ensemble des transitions successives entre les états globaux S(t) successifs résultant de l'exécution d'une application sur l'architecture. Dans notre approche ce graphe est construit dynamiquement par le PMU. Nous détaillons sa construction dans ce paragraphe.

Pour illustrer la construction du graphe de l'historique, nous considérons l'exemple de l'architecture présentée sur la Figure V.1 qui est composée de trois blocs IP. Nous observons les changements des états de ces blocs durant l'exécution d'une application simple. Les deux états S(t_1)=(*Idle*, Actif, Inactif) et S(t_2)=(Actif, Actif, *Idle*) des blocs IP se produisent selon les états des tâches prêtes à s'exécuter et la politique d'ordonnancement. À l'état S(t_1), chacun des blocs IP garde son état stable pendant un intervalle de temps égal à $\Delta_{S(t_1)}= t_2 - t_1$. Supposons dans un premier temps que l'état S(t_2) est le seul successeur de l'état S(t_1), donc la prochaine fois où le système entre dans l'état S(t_1), nous pouvons prédire que l'état prochain du système devrait être l'état S(t_2) et que l'état S(t_1) restera stable durant un temps égal à $\Delta_{S(t_1)}$. Dans une première approximation, on peut considérer que lorsque le système entre dans l'état S(t_1) et que la période de stabilité S(t_1) est supérieure au *break-even-time* t_{be,x_0} de l'unité x_0, le bloc x_0 peut être mis en mode *power gated*. Dans le cas contraire ($\Delta_{S(t_1)} < t_{be,x_0}$), uniquement l'état de *clock gated* peut être appliqué afin de réduire la consommation dynamique de puissance du bloc .

Power Mode	***Run***	***Sleep***	***Stop***
mA	93 @168Mhz	50 @168Mhz	0.45
Wake-up (ns)	NA	1000	13000

Tableau V.1 – États de puissance de CPU ARM Cortex M4 dans STM32F405xx

Généralement, les cœurs de processeurs possèdent plus d'un état de basse consommation. Un exemple dans le tableau V.1 donne les caractéristiques d'un sous-ensemble des états de puissance du CPU ARM Cortex M4 dans la puce STM32F405xx [dat13]. Dans le cas où il y

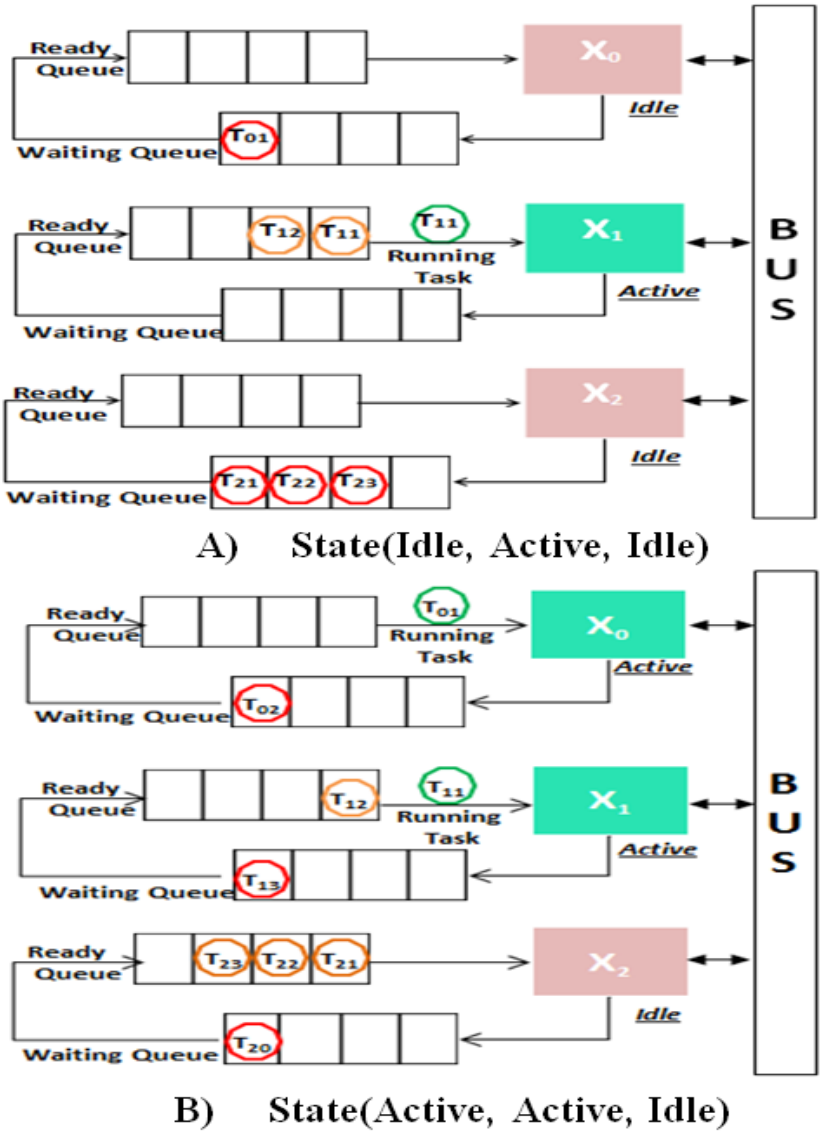


Figure V.1 – Exemple des états

a plusieurs états de puissance pour un bloc IP x_0, nous sélectionnons le plus efficace en tenant compte des valeurs de $\Delta_{S(t_1)}$ et de t_{be,x_0}.

Pour construire le graphe de l'historique des états, le PMU reçoit de chaque bloc IP, ou de chaque *Power Domain*, un signal de notification qui indique un changement de l'état actif à l'état inactif et inversement. Il reçoit également des événements de réveil venant de la partie matérielle ou logicielle utilisés pour activer les blocs IP inactifs (par exemple, à partir des pilotes logiciels du système ou du noyau Linux avec la fonction *Cpu-Idle*()). Ces signaux sont appelés les signaux des états fonctionnels. Chaque fois qu'au moins un de ces signaux est déclenché, le

PMU met à jour le graphe de l'historique des états.

Soit S(t_m) = ($s_0(t_m)$, $s_1(t_m)$, ..., $s_{n-1}(t_m)$) l'état global des n blocs IP observés au temps t_m résultant du déclenchement d'au moins un signal d'état fonctionnel. Soit S(t_0), S(t_1), ..., S(t_{m-1}) l'ensemble des états précédents déjà insérés dans le graphe de l'historique. Si S(t_m) est déjà présent dans le graphe de l'historique avec S(t_m) = S(t_{m-k}), S(t_{m-k}) est considéré alors comme le successeur le plus récent de l'état S(t_{m-1}). Dans le cas contraire, un nouvel état S(t_m) est inséré dans le graphe de l'historique avec S(t_{m-1}) comme prédécesseur. Dans les deux cas, la période de stabilité de S(t_{m-1}) est alors mise à jour avec la valeur $\Delta_{S(t_{m-1})} = t_m - t_{m-1}$. Cet algorithme de construction du graphe de l'historique des états est décrit dans l'algorithme 1.

Si S(t_m) est un nouvel état, sa période de stabilité est définie à 0 (ligne 11). Sinon, la période de stabilité de son prédécesseur est mise à jour (ligne 6) comme suit : nous multiplions la fréquence de fonctionnement courante du bloc IP par la différence des instants d'occurrence des signaux fonctionnels et divisée par la fréquence maximale qui est définie comme une constante. Ainsi dans la ligne 6 les périodes de stabilité sont enregistrées relativement à la fréquence maximale de manière à être indépendant de toute stratégie DVFS exécutée au niveau du système d'exploitation ou d'un *middleware*.

Algorithm 1: Algorithme de la construction de graphe de l'historique

```
1:  for each state S(t_m) do
2:    if S(t_m) ∈ {S(t_0), S(t_1), ..., S(t_{m-1})} then
3:      /* S(t_m) was previously encountered, thus S(t_m) has          */
        /* at least one successor                                     */
4:      Let S(t_{m-k}) = S(t_m), 0 ≤ k ≤ m − 1;
5:      Update S(t_{m-k}) as the most recent successor of S(t_{m-1});
6:      Δ_{S(t_{m-1})} = (t_m − t_{m-1}) * Current_Frequency / Max_Frequency;
7:    else
8:      /* S(t_m) is a new state                                      */
9:      Insert S(t_m) in the graph as a successor of S(t_{m-1});
10:     Set S(t_m) as the most recent successor of S(t_{m-1})
11:     Δ_{S(t_m)} = 0 /* as S(tm) has no successor its stable period is set to 0   */
12:   end if
13: end for
```

Sur la Figure V.2, un graphe de l'historique résultant de l'exécution du graphe de tâches du *H264 Encoder* sur une architecture composée de quatre blocs IP est illustré. L'exemple du *H264 Encoder* est décrit dans le paragraphe 3.2.2.

Le but est ensuite d'utiliser le graphe de l'historique et les relations entre ces états pour

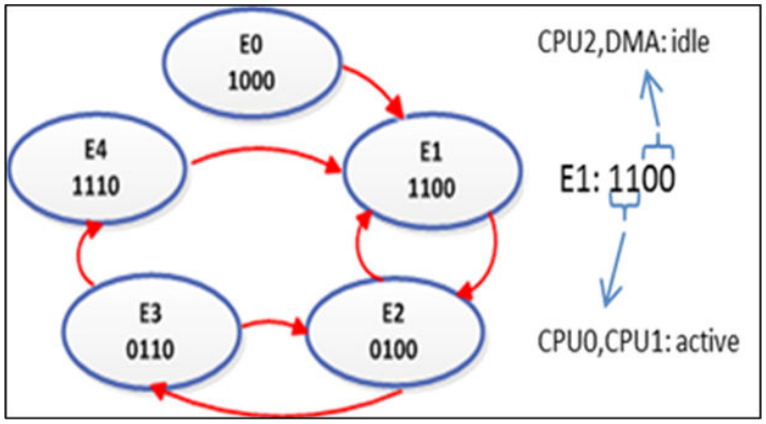


Figure V.2 – Graphe de l'historique du cas d'étude *H264 Encoder* 3.2.2

prédire l'état du système à partir d'un état courant.

2.2 Sélection du mode de puissance

À partir du graphe de l'historique, nous souhaitons prédire le mode de puissance le plus adapté pour le système en fonction du futur proche et ce afin d'éviter que des pénalités de temps et d'énergie ne viennent affecter les performances ou les gains en énergie. Ceci peut se faire en anticipant pour chaque bloc IP le changement d'état de puissance d'un mode basse consommation à un mode actif de telle sorte que ce changement soit effectif au moment où l'activité fonctionnelle du bloc doit être déclenchée. Ainsi, ce temps de réveil est masqué par le temps d'inactivité du bloc IP. Cette prédiction de mode de puissance est accomplie dynamiquement par le PMU à chaque instant $\mathrm{S}(t_m)$ lorsque l'état $s_j(t_m)$ du bloc x_j correspond à un état fonctionnel inactif.

Afin de masquer les pénalités de temps et d'énergie, l'anticipation des modes de puissance des blocs IP ne peut être réalisée que seulement si la période de stabilité de l'état actuel $\mathrm{S}(t_m)$ peut être estimée. Cette estimation n'est possible que si l'état $\mathrm{S}(t_m)$ a au moins un successeur dans le graphe de l'historique. Dans le cas contraire, appliquer le *power gating* engendrerait des risques de dégradation de performances. Toutefois, la fonction de *clock gating* peut être appliquée pour chaque bloc IP x_j inactif dans $\mathrm{S}(t_m)$ puisque le temps de pénalité pour désactiver et réactiver l'horloge peut être négligé généralement. Conséquemment, nous supprimons la composante dynamique de la puissance dans cet intervalle de temps pour ces blocs IP inactifs.

En comparaison avec ce que nous avons indiqué précédemment, nous limitons à un nombre d'états limité $S(t_{m-p}), S(t_{m-p+1}), S(t_{m-p+2}), S(t_{m-1})$ l'ensemble des états globaux les plus ré-

cents dans le graphe de l'historique en prenant en compte l'impact du processus du vieillissement qui sera détaillé dans 2.3. Cette fenêtre sur le nombre d'états considérés a pour but de borner les temps de recherche de l'état successeur et également de faire abstraction d'états anciens qui pourraient être liés à une activité globale issue par exemple de l'exécution d'une précédente application.

Algorithm 2: Algorithme de la prédiction du mode de puissance

1: **for each state** S(t_m) **do**
2: **if** $S(t_m) \in \{S(t_{m-p}), S(t_{m-p+1}), ..., S(t_{m-p+2}), S(t_{m-1})\}$ **then**
3: /* S(t_m) has at least a successor in the history graph */
4: Let S(t_{m-k}) = S(t_m) / $m-p \leq k \leq m-1$;
5: S(t_{m-k+1}) is then the successor of S(t_m) corresponding to the most recently used outgoing edge from S(t_{m-k})in history graph
6: $\Delta_{S(t_m)} = (t_{m-k+1} - t_{m-k}) * \frac{Max\_Frequency}{Current\_Frequency}$;
7: S(t_{m-k+2}) is successor of S(t_{m-k+1});
8: $\Delta_{S(t_{m-k+1})} = (t_{m-k+2} - t_{m-k+1}) * \frac{Max\_Frequency}{Current\_Frequency}$;
9: **for each** $x_j \in$ S(t_m) /$s_j(t_m)$=*idle* **do**
10: $t_{idle} = \Delta_{S(t_m)}$;
11: **if** $S(t_{m-k+1}) = idle$ **then**
12: $t_{idle} = t_{idle} + \Delta_{S(t_{m-k+1})}$
13: **end if**
14: Select the most efficient power mode PM_{x_j} for x_j such that $t_{be,x_j} \leq t_{idle}$;
15: **if** $PM_{x_j} \neq clockgating$ **then**
16: schedule a wake-up event for x_j at time $t_m + t_{idle} - t_{be,x_j}$
17: **end if**
18: **end for**
19: **else**
20: /* $S(t_m)$ is a new state */
21: $\Delta_{S(t_m)} = 0$
22: **for each** $x_j \in$ S(t_m) /$s_j(t_m)$=*idle* **do**
23: apply clock gating to x_j
24: **end for**
25: **end if**
26: **end for**

L'algorithme de la prédiction du mode de puissance des blocs IP est illustré sur l'algorithme 2. La ligne 4 permet de définir le successeur direct de l'état actuel S(t_m) et la ligne 7 permet de définir le successeur de ce successeur de manière à augmenter la période (ligne 12) sur laquelle un mode de puissance efficace peut être appliqué aux blocs IP. Nous notons que la gestion du DVFS peut être appliquée au niveau du système d'exploitation ou du *middleware* en

estimant la période de stabilité de l'état, indépendamment de la fréquence de fonctionnement appliquée au bloc IP (ou au *Clock Domain*). Ceci est réalisé à la ligne 6 de l'algorithme 2 en procédant de manière inverse par rapport à la ligne 6 de la l'algorithme 1.

Á la ligne 14, le mode de puissance est choisi de manière à maximiser les économies d'énergie. En effet, quel que soit la période de stabilité de l'état $\Delta_{S(t_m)}$ (même si elle est nulle) un *clock gating* est appliqué sur les blocs IP inactifs. Mais le *power gating* ne peut être appliqué que si la période de la stabilité de l'état $\Delta_{S(t_m)}$ est supérieure au temps de pénalité. Dans ce cas, pour éviter les pénalités sur la performance, un événement de réveil pour le bloc IP x_j est fixé à l'instant $t_{wake-up}$. Cet instant du réveil est défini comme suit :

$$t_{wake-up} = t_m + t_{idle} - t_{be,x_j} \tag{V.2}$$

En fait, à $t_m + t_{idle}$ le bloc x_j doit être actif et nous prenons en compte le temps nécessaire (t_{be,x_j}) pour que le bloc x_j bascule du mode inactif vers le mode actif. Toutefois, en cas d'erreur de prédiction, l'état du bloc IP est modifié selon l'occurrence de la notification fonctionnelle de réveil. Ce cas engendre des pénalités de temps et d'énergie résultantes des mauvaises prédictions.

2.3 Processus de vieillissement du graphe de l'historique

Le comportement des applications peut être irrégulier et la demande en puissance de calcul peut varier pendant l'exécution ce qui conduit à activer/désactiver différents blocs IP en fonction des besoins fonctionnels. En outre, différentes applications avec des complexités variables peuvent être exécutées en séquence sur le même système ce qui implique alors divers besoins en ressources. Ceci peut perturber la sélection des modes de puissance adaptés si le graphe de l'historique contient des états non significatifs vis-à-vis de l'application en cours d'exécution. Par conséquent, le graphe de l'historique doit évoluer pour tenir compte des changements significatifs qui interviennent au niveau fonctionnel suivant la dynamique de la charge induite par les applications. Pour ce faire, nous avons mis en place un mécanisme de vieillissement du graphe de l'historique dans le PMU. Notre PMU étant générique, son comportement doit être indépendant des applications exécutées sur le système et donc, le mécanisme de vieillissement doit être lui-même indépendant du comportement de l'application.

En se basant toujours sur le postulat que pour un état courant, l'état successeur le plus récent est le plus probable à s'exécuter à nouveau, nous considérons tout d'abord une méthode simple pour implémenter un mécanisme de vieillissement : un processus périodique de période Δp supprime le ou les état(s) le(s) plus ancien (s) du graphe de l'historique. À cet effet, un

compteur d'événements, noté C dans le PMU, est incrémenté à chaque instant t_m où un signal de l'état fonctionnel est déclenché. Un registre $u_{S_{(t_m)}}$ associé à l'état $S(t_m)$ est mis à jour avec le contenu du compteur C à chaque fois que le système atteint cet état. À chaque instant périodique $K * \Delta p$, la valeur du registre $u_{S_{(t_m)}}$ est copiée dans un deuxième registre $b_{S_{(t_m)}}$. Si pendant une période Δp le système ne passe jamais à travers l'état $S(t_m)$, les contenus des deux registres restent inchangés ($u_{S_{(t_m)}} = b_{S_{(t_m)}}$). Nous pouvons déduire que l'état $S(t_m)$ est inutilisé pendant la période Δp alors nous pouvons le considérer comme ancien et le retirer du graphe de l'historique. Nous supprimons également du graphe tous les successeurs de l'état $S(t_m)$ et qui n'ont pas d'autres prédécesseurs que $S(t_m)$ car ils ne peuvent pas être atteints a priori du fait que l'état $S(t_m)$ est retiré du graphe.

L'utilisation du processus périodique de période Δp fixe comme mécanisme de vieillissement a un inconvénient. En effet, avec une période Δp longue à l'égard du temps d'exécution de l'application, certains états restent inutilisés dans le graphe mais peuvent polluer les décisions de prédiction au moment de la sélection du mode de puissance. À l'inverse, avec une durée trop courte, des états significatifs peuvent être retirés ce qui limite l'efficacité de la stratégie de *power management*. Pour résoudre ce problème, un mécanisme d'adaptation de la valeur de période Δp pourrait être envisagé, mais ce mécanisme ne serait pas adapté ou complexe à mettre en œuvre lorsque des changements importants se produisent dans le système (par exemple, des migrations des tâches, activation d'une nouvelle application, etc.). Par conséquent, pour réaliser ce processus de vieillissement, nous avons également considéré l'utilisation d'un événement résultant de deux mauvaises prédictions de l'état $S(t_m)$ successives rencontrées dans le PMU. Cet événement asynchrone permet d'être indépendant de la dynamique des applications en particulier lorsque le système bascule entre deux ou plusieurs applications ou quand une migration de la tâche se produit (par exemple, ceci est effectué dans la plateforme big.LITTLE ARM pour tenir compte d'effets thermiques [MPV+13b]). Nous notons par HG_PM cette approche de gestion de puissance basée sur l'historique des états globaux du système.

3 Application sur des cas d'études

Dans le but d'évaluer notre stratégie de gestion de puissance, nous considérons deux exemples d'application. Le premier est fourni dans l'environnement *Platform Architect* du *Synopsys* [3] et se compose d'un sous-ensemble d'un codeur H264 [ZAAJ09]. Le deuxième exemple noté *Video_Capture* décrit un ensemble d'actions pour l'acquisition d'une séquence d'images à partir d'une interface d'appareil photo et d'un flux audio à partir d'une interface de microphone. Les

deux cas d'études sont décrits par un graphe de tâches et sont projetés (*mapping*) sur un modèle d'une architecture décrite en SystemC-TLM et composée de trois processeurs notés CPU0, CPU1 et CPU2, d'une unité DMA et d'une mémoire organisée en cinq bancs. Ces blocs IP sont reliés par un bus commun. Ce *mapping* est indiqué sur la Figure V.3 et la Figure V.4 où le nombre du bloc IP affecté à une tâche est donné en haut à droite de chaque rectangle symbolisant une tâche. Nous notons aussi dans les deux Figures, le nombre de cycles nécessaires pour exécuter chaque tâche et la taille des données transférées en lecture (Rd) depuis la mémoire ou en écriture (Wr) vers la mémoire.

Dans la suite, nous décrivons tout d'abord l'outil *Platform Architect* ainsi que la stratégie utilisée par le PMU développé dans cet outil par *Synopsys* pour effectuer une évaluation rapide de la consommation de puissance qu'il est possible d'atteindre. Ensuite, nous présentons les cas d'études ainsi que les résultats obtenus en appliquant l'approche proposée.

3.1 Présentation de l'environnement *Platform Architect* de *Synopsys*

L'environnement *Platform Architect with Multicore Optimization* (MCO) offre des outils et des méthodes basés sur une simulation SystemC-TLM pour la conception, l'analyse de performance et l'optimisation de systèmes multi cœurs et d'architectures de type SoC.

Avec cet environnement, les concepteurs peuvent explorer différentes solutions de partitionnement matériel-logiciel et la configuration de l'infrastructure de SoC, en particulier l'interconnexion globale et la mémoire et ce pour obtenir au niveau d'une modélisation abstraite la performance souhaitée et une idée du coût du système induit.

L'analyse de la performance avec *Platform Architect* MCO se base sur une simulation au niveau transactionnel permettant d'obtenir un temps de réponse rapide. L'outil propose une visualisation des résultats utilisable par les concepteurs pour leur permettre d'identifier les possibilités d'amélioration et ainsi de prendre les décisions ad-hoc pour tendre vers une solution optimisée. Cette analyse est faite à travers un ensemble de vues et de traces.

Les composants de l'architecture sont représentés par des instances de VPU (*Virtual Processing Unit*) dans la plateforme SystemC-TLM. En effet, un VPU est un modèle abstrait d'un composant de l'architecture matérielle, il peut s'agir d'un CPU ou d'un périphérique ou d'un composant matériel spécifique. Il dispose de différents paramètres génériques qui permettent de le particulariser. Chaque VPU a un ordonnanceur qui sélectionne la tâche prête à être exécuter selon une stratégie *Round-Robin*. Dans une architecture modélisée dans *Platform Architect* à partir de VPU et contrôlée en puissance, nous trouvons un VPU spécifique représentant un

PMU qui implémente une stratégie de gestion de puissance basique. Cette stratégie consiste à réduire la tension et la fréquence d'un bloc IP dès que ce bloc entre dans un état de repos (*Idle*). Nous appelons ce PMU *Basic_PM*. Nous notons que si tous les temps de pénalités pour changer de mode de puissance sont nuls (alors $T_{be} = 0$), le *Basic_PM* est alors optimal puisque dès qu'un temps d'inactivité se produit, la consommation d'énergie statique et dynamique sont à la fois réduites par utilisation du mode de puissance le plus efficace et ce sans aucune pénalité de temps et d'énergie. Nous comparons ainsi notre stratégie avec cette technique de *power management* basique fournie dans *Platform Architect*. Deux modes de puissance sont considérés dans cette étude pour les blocs IP où on suppose que la fréquence d'horloge et la tension d'alimentation sont toutes les deux ajustables par bloc IP (chaque bloc IP est à la fois un *Power Domain* et un *Clock Domain*) :

- mode *Stop* : Vdd = 0 V, Fréquence = 0 ;
- mode *Run* : Vdd = 1.1V, Fréquence = 400 MHz.

Nous définissons le temps de pénalité t_{pdu} par la somme des délais nécessaires pour passer du mode Stop au mode *Run* :

$$t_{pdu} = Power\_up\_delay + Power\_down\_delay \tag{V.3}$$

Dans l'étude réalisée ici, nous avons considéré qu'en mode *Run*, la puissance dynamique par CPU peut atteindre jusqu'à 0.2 W tandis que la puissance statique par CPU est 0.066W. Nous notons que les OPP (V, F) définis pour les modes de puissance sont communs à tous les blocs IP mais considérer des OPP différents par bloc IP ne pose pas de difficulté.

3.2 Application de HG_PM sur les cas d'étude *Video_Capture* et *H264 Encoder*

Le graphe de tâches d'un cas d'étude est décrit dans *Platform Architect* sous forme d'un fichier CSV (*Comma Separated Values*) qui regroupe les informations sur les tâches, leurs dépendances, leurs contraintes d'activité, etc. La sémantique d'un graphe de tâches décrit selon le format CSV est similaire à celle d'un graphe de données synchrone (SDF) introduite dans [LM87] où à chaque activation, une tâche consomme des jetons sur chacun de ses arcs d'entrée et produit des jetons sur chacun de ses arcs sortants.

3.2.1 Présentation du cas d'étude *Video_Capture*

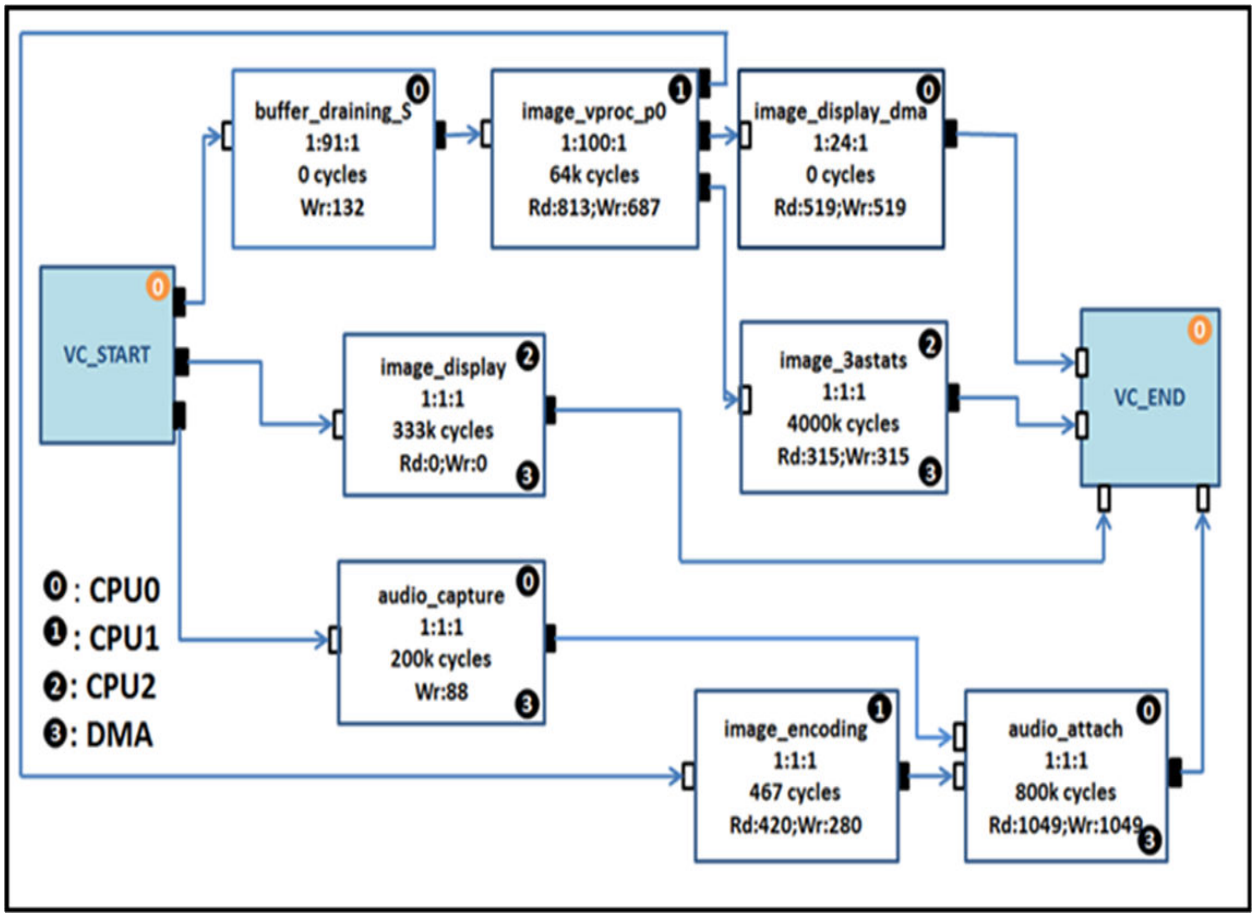


Figure V.3 – Graphe de tâches de *Video_Capture*

Video_Capture décrit un ensemble d'actions pour acquérir une séquence d'images et un flux audio. Ces actions comprennent des tâches de transferts de données, de stockage de données, de codage d'image et de traitement. Des images successives produites par la caméra sont traitées en utilisant un processus pipeline permettant ainsi à différentes tâches relatives à une image de s'exécuter en parallèle avec des tâches associées à des images suivantes. Cette approche pipeline accélère le débit des images traitées, mais impose que la conception de l'architecture soit capable de supporter l'activation parallèle des tâches impliquées dans l'exécution des traitements de chaque image. Dans cette étude, un débit de 60 images par seconde est visé ce qui correspond à l'acquisition d'une nouvelle image toutes les 16.6 ms.

Le graphe global de tâches de *Video_Capture* est représenté dans la Figure V.3. Ce graphe de tâche est hiérarchique. Chaque tâche peut être composée d'au plus cinq sous-tâches. Ces sous-

tâches sont séquentielles au sein de la tâche. La notation 1 : x : 1 indique que les sous-tâches internes sont exécutées x fois (éventuellement en mode pipeline) alors que la tâche consomme un jeton sur son arc entrant et produit un jeton sur son arc sortant.

3.2.2 Présentation du cas d'étude *H264 Encoder*

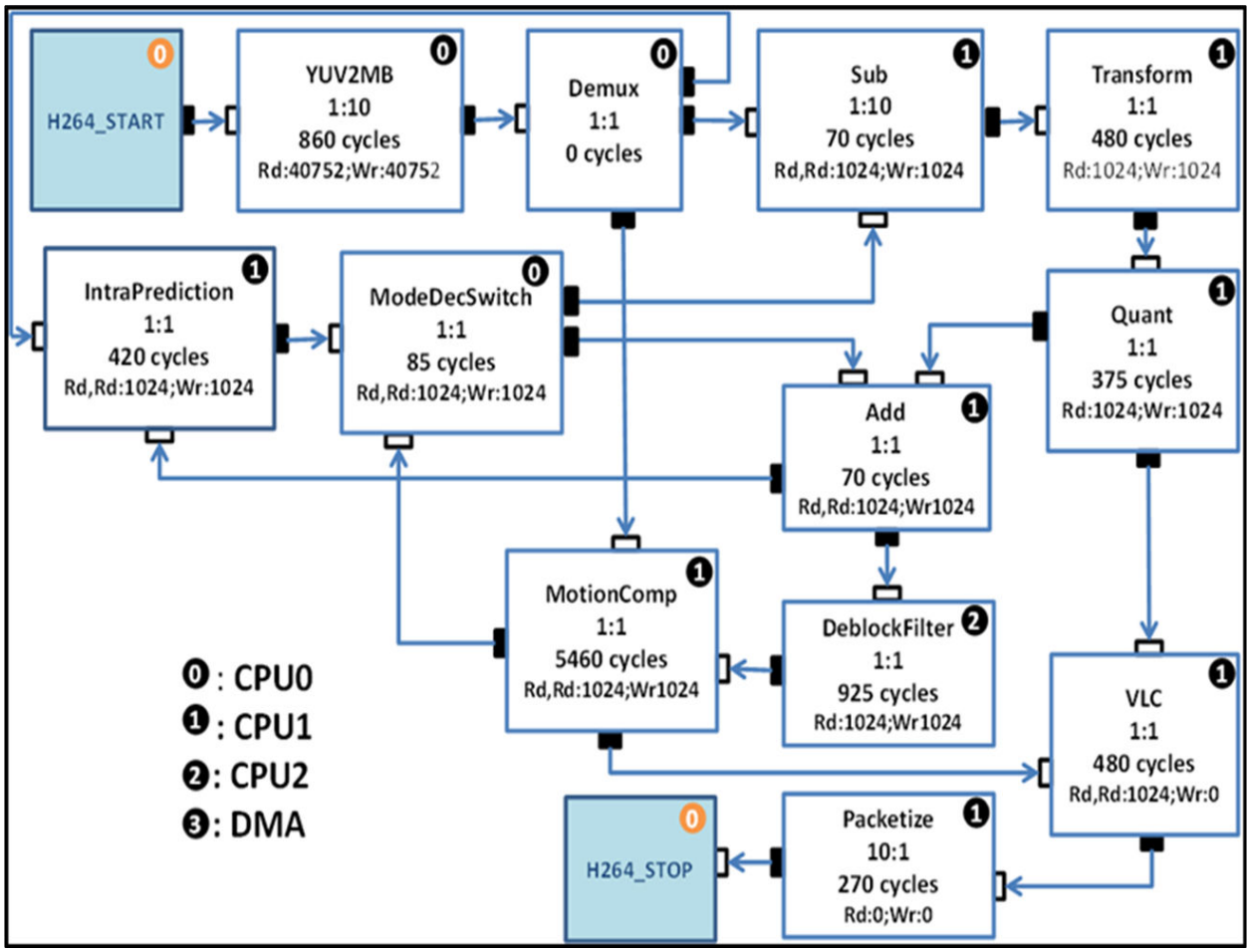


Figure V.4 – Graphe de tâches de *H264 Encoder*

L'application *H264 Encoder* prend en entrée une séquence de trames, les divise en un certain nombre de macro-blocs (par exemple, 16x16 pixels) et génère un *bitstream* compressé. Comme dans l'encodage JPEG pour les images fixes, la compression principale est réalisée par une opération de DCT (*Discrete Cosine Transform*). Typiquement, la DCT transforme les blocs (8x8 ou 4x4 pixels) de valeurs de *greyscale* dans un domaine de fréquence. Ces valeurs peuvent être ensuite compressées par quantification et un codage entropique. Un macrobloc (16x16

pixels) peut être divisé en quatre blocs (8x8 pixels) d'informations de luminance (Y) et de deux blocs (8x8 pixels) d'informations de chrominance (U, V, respectivement). La compression vidéo H.264 est basée sur la prédiction. En effet, l'opération DCT n'est pas effectuée directement sur le macrobloc (MB) mais plutôt sur un macro-bloc prédit. Le graphe global de tâches de *H264 Encoder* est représenté dans la Figure V.4. Nous notons que la notation x : y sur une tâche indique que à chaque exécution, x jetons sont consommés sur son arc entrant et y jetons sont produites sur son arc sortant.

3.2.3 Résultats de l'application de HG_PM

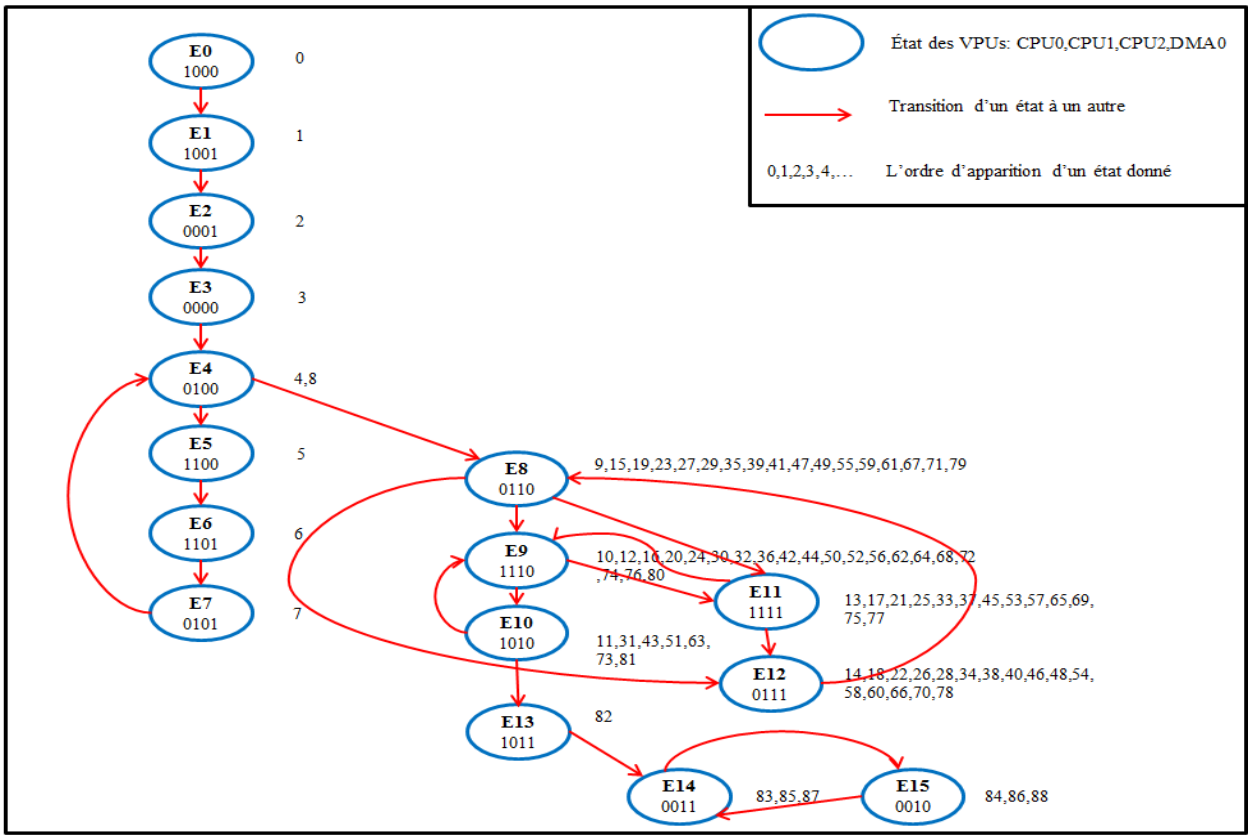
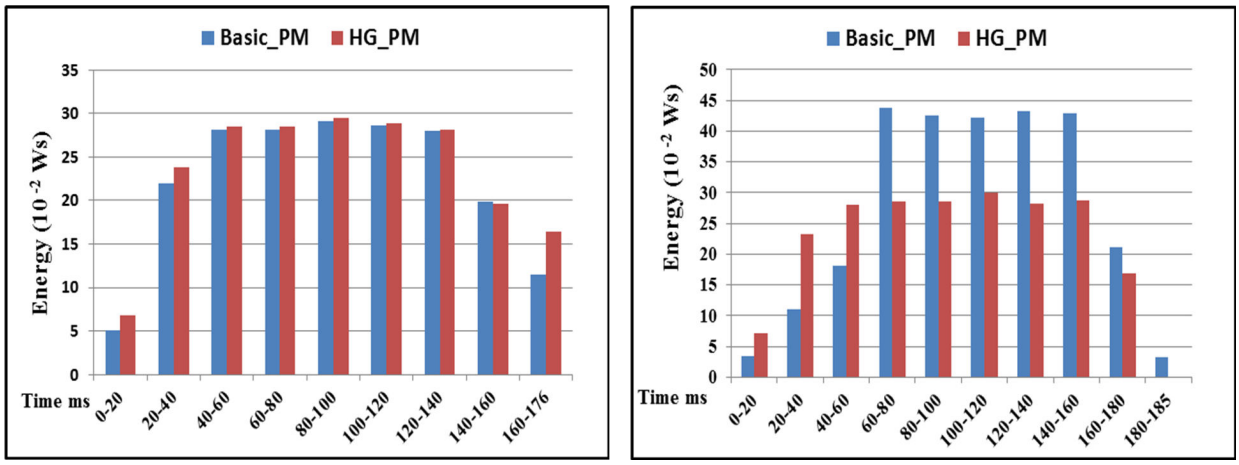


Figure V.5 – Graphe de l'historique *Video_Capture*

Le graphe de l'historique résultant de l'exécution de *Video_Capture* sans exécuter le processus de vieillissement est illustré sur la Figure V.5. Les numéros d'événement des signaux de l'état fonctionnel associé à chaque état S(t) sont annotés aussi sur la figure. Ce graphe de

l'historique est composé de seize états. Les quatre blocs IP passent donc par tous ces états mais après l'événement numéro 8, les états E0 à E7 ne sont plus utilisés. Ce comportement est dû au recouvrement des huit premières instances du graphe de tâches exécutées avec un délai d'activation entre deux instances de 16.6 ms. De plus, après l'événement numéro 82, seulement les états E14 et E15 sont utilisés pour l'exécution de la dernière instance.

L'exécution du graphe de tâches *H264 Encoder* permet de construire le graphe de l'historique illustré sur la Figure V.2. Seuls cinq états sont utilisés dans cet exemple sur le total de seize états possibles.

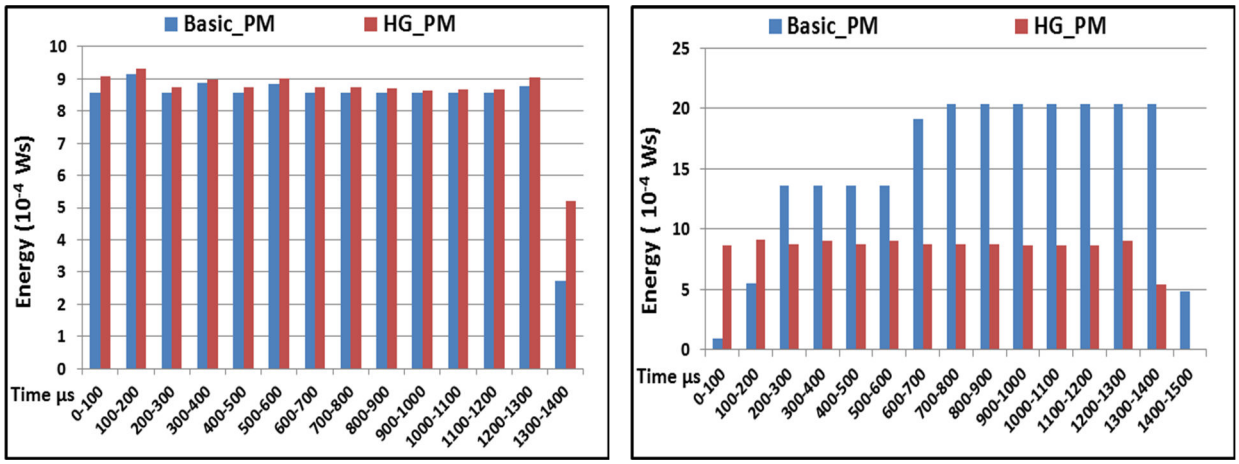


(a) Temps de pénalité = 1.8 μs (b) Temps de pénalité = 180 μs

Figure V.6 – Consommation d'énergie *Video_Capture*

En vue d'évaluer la stratégie basée sur l'analyse des états fonctionnels, nous avons comparé l'énergie consommée des graphes de tâches en appliquant la stratégie de gestion de puissance HG_PM avec celle consommée en appliquant la stratégie de gestion de puissance *Basic_PM*. La Figure V.6 et la Figure V.7 montrent l'énergie totale consommée par intervalle de temps des graphes de tâches *Video_Capture* et *H264 Encoder* respectivement en utilisant les deux modes de puissance *Stop* et *Run* décrits ci-dessus. Nous considérons deux différents temps de pénalité 1.8 μs (les Figures V.6(a) et V.7(a)) et 180 μs (les Figures V.6(b) et V.7(b)).

Comme nous pouvons l'observer au début de l'exécution, l'énergie consommée en appliquant HG_PM est plus élevée que celle avec *Basic_PM*. Cela est dû en premier lieu à la phase d'apprentissage initial de la procédure de prédiction. En deuxième lieu, dans le cas où le *Basic_PM* est appliqué, les blocs IP ne sont pas initialement en mode *Stop* mais ils sont en mode *Run* en



(a) Temps de pénalité = 1.8 μs (b) Temps de pénalité = 180 μs

Figure V.7 – Consommation d'énergie *H264 Encoder*

appliquant HG_PM. Avec un temps de pénalité faible (1.8 μs, les Figures V.6(a) et V.7(a)) le *Basic_PM* fournit de meilleurs résultats en terme de consommation d'énergie. Dans ce cas, la différence d'énergie entre le *Basic_PM* et HG_PM est due aux mauvaises prédictions par l'HG_PM lorsque la condition de *break even time* n'est pas satisfaite. Ces mauvaises prédictions dissipent de l'énergie qui n'est pas compensée en changement du mode de puissance en utilisant le *Basic_PM*. Bien que lorsque la pénalité de temps soit réglée à 180 μs, nous pouvons voir sur les Figures V.6(b) et V.7(b) une réduction jusqu'à 50% de l'énergie totale consommée et une diminution du temps de latence jusqu'à 10% les en appliquant la stratégie HG_PM.

3.3 Application de HG_PM sur un graphe de tâches composé de *H264 Encoder & Video_Capture*

Dans le but d'évaluer le comportement de notre PMU, y compris le processus de vieillissement, nous fusionnons les deux graphes de tâches décrits ci-dessus *H264 Encoder* et *Video_Capture* pour construire un graphe de tâches composé (concaténation des deux graphes) qui impose au système de basculer entre deux applications différentes. Nous comparons l'énergie consommée par ce graphe composé avec celle consommée d'une part par le graphe de tâches *H264 Encoder* et d'autre part par le graphe *Video_Capture*.

Time Penalty (μs)	*Aging Period* (ms)	Average Energy ($10^{-}2\ Ws$)		
		H264 Encoder	*Video_Capture*	*Task Composite*
0.18	0.1	0.130	22.48	22.53
	1	0.130	22.36	22.39
	10	0.130	22.07	22.08
18	0.1	0.130	22.50	22.59
	1	0.131	22.48	22.59
	10	0.131	22.46	22.57
180	0.1	0.133	22.48	22.60
	1	0.133	22.47	22.59
	10	0.133	22.46	22.57

Tableau V.2 – Résultats avec différents temps de pénalité et différentes périodes de vieillissement

Le tableau V.2 montre les résultats en tenant compte de trois périodes Δp d'activation du processus de vieillissement (0,1ms, 1ms et 10 ms) et de trois pénalités de temps différentes (0.18 μs , 18 μs et 180 μs). Les colonnes *H264 Encoder* et *Video_Capture* illustrent les consommations d'énergie individuelles pour chaque graphe. La colonne *Task Composite* donne les résultats du graphe de tâches composé *H264 Encoder & Video_Capture*. Nous observons des différences mineures dans l'application du processus de vieillissement. Ceci est expliqué par la nature itérative de l'application et par notre stratégie de prédiction basée sur l'utilisation d'états récents qui ne sont pas éliminés par le processus de vieillissement. Cependant, les résultats sont meilleurs lorsque la période de vieillissement est plus faible que le temps d'exécution de l'application (0.1 ms pour *H264 Encoder*) mais cette période ne doit pas être trop petite pour ne pas retirer les états récents utiles (par exemple, 0.1 ms ou 1 ms pour *Video_Capture*). Par conséquent, la période de vieillissement de 10 ms apparaît ici comme le meilleur compromis pour le graphe de tâche composé puisque *Video_Capture* est le graphe le plus contributif à la consommation d'énergie totale.

Comme illustré dans le tableau V.3, le nombre de mauvaises prédictions est plus faible lors de l'application du processus de vieillissement (104) avec une période de 10 ms par rapport aux mauvaises prédictions produites (106) sans exécuter le processus de vieillissement. Le nombre total d'états créés dans le graphe de l'historique peut augmenter jusqu'à 29 (par rapport à 16) lorsque le processus de vieillissement est activé avec une période de 10 ms. Ceci s'explique par soit une suppression inopportune d'états qui aurait pu permettre une sélection d'un mode de

puissance efficace soit une suppression d'un état obsolète car relatif à la première application mais reconstruit dans la deuxième application avec de nouveaux successeurs/prédécesseurs.

	H264 Encoder		***Video_Capture***		***Task Composite***	
# State without Aging	5		16		16	
# Misprediction without Aging	68		31		106	
Aging event	Δp=0.1 ms	Dble-Miss	Δp=10ms	Dble-Miss	Δp=10ms	Dble-Miss
# State with Aging	6	7	26	45	29	51
# Misprediction with Aging	68	69	35	22	104	94

Tableau V.3 – Nombres des états et des mauvaises prédictions

Si nous considérons que le processus de vieillissement est activé seulement lorsque deux mauvaises prédictions successives se produisent (*Dble_Miss*), nous notons que 51 états sont insérés dans le graphe de l'historique du graphe de tâches composé *H264 Encoder & Video_ Capture* et 94 mauvaises prédictions sont rencontrées. Dans ce cas, la consommation totale d'énergie est de $22 * 10^{-2} Ws$. En comparant avec l'application d'un vieillissement périodique, le nombre d'états créés est plus important mais le nombre de mauvaises prédictions est réduit. Ceci illustre que certains états non pertinents sont correctement éliminés par le processus de vieillissement lorsque le système bascule entre les deux applications.

4 Conclusion

Dans ce chapitre, nous avons proposé une stratégie dynamique de gestion de puissance pour les systèmes sur puce. Cette stratégie est basée sur un graphe de l'historique d'états fonctionnels globaux des blocs IP de l'architecture. Un état fonctionnel global est la conséquence de la composition du *mapping* et de l'ordonnancement des tâches de l'application sur les blocs IP. Par conséquent, l'analyse du graphe de l'historique fournit des informations pertinentes pour la prise de décision de gestion de puissance et il n'y a pas besoin d'informations supplémentaires pour appliquer cet algorithme, en particulier venant des niveaux application et OS. Afin d'améliorer la prédiction des décisions de gestion de puissance, nous avons ajouté une

technique de vieillissement du graphe de l'historique dans le mécanisme de construction du graphe de manière à éliminer les états non pertinents induits par exemple lors d'un changement d'application exécutée ou de migration de tâches. Les résultats de nos cas d'études montrent l'impact important sur les économies d'énergie sans affecter significativement la performance.

Chapitre VI

Conclusion générale et perspectives

Pour clore la présentation de notre travail de thèse, nous présentons tout d'abord une synthèse de nos contributions pour la modélisation au niveau transactionnel de l'architecture et du contrôle relatifs à la gestion d'énergie de systèmes sur puce puis, nous exposons des exemples de travaux futurs qui compléteraient le travail réalisé.

1 Résumé des contributions

La conception d'un système sur puce doit prendre en considération plusieurs contraintes afin que le système soit utilisable dans de bonnes conditions. La consommation d'énergie est une contrainte comptée parmi les plus influentes sur l'efficacité du système du fait qu'elle agit directement sur l'autonomie de ce dernier lorsque celui-ci est alimenté par batterie. Pour répondre à cette contrainte, l'étude de la consommation de puissance et l'intégration d'une stratégie particulière doit se situer dans une étape précoce du flot de développement d'un système intégré.

L'objectif principal que nous nous sommes fixé est de proposer des solutions permettant la conception par modélisation au niveau transactionnel d'architecture et du contrôle des systèmes sur puce dans le but d'atteindre la plus grande économie d'énergie possible. Dans la suite, nous résumons ces solutions en trois contributions principales.

1.1 Approche de gestion d'horloge au niveau transactionnel

Nous avons étudié une approche de modélisation au niveau transactionnel pour les systèmes sur puce qui associe à un modèle fonctionnel SystemC-TLM une description d'une structure de gestion d'un arbre d'horloge décrit au même niveau d'abstraction. Cette structure développée suivant le principe de séparation des préoccupations fournit à la fois l'interface pour la gestion de puissance des composants matériels et pour le logiciel applicatif. L'ensemble des modèles

développés est rassemblé dans une librairie *ClkARCH*. Cette librairie facilite l'intégration d'une architecture en *Clock Domain* dans un modèle SystemC-TLM. Pour associer un modèle de gestion d'un arbre d'horloge à un modèle fonctionnel, nous avons proposé une méthodologie en trois étapes : la spécification du *clock intent*, la modélisation de la stratégie de gestion, et la simulation globale de l'architecture. Une étape de vérification en simulation est aussi considérée en parallèle à ces étapes en se basant sur des contrats génériques. En effet, les propriétés liées principalement à la structure de la gestion d'arbre d'horloge et ses effets sur le fonctionnement initial du modèle fonctionnel sont définies sous forme des contrats. Lors de l'application du flot de notre méthodologie, ces contrats sont vérifiés progressivement sous la forme de types d'assertions.

1.2 Extension du standard IP-XACT pour supporter une stratégie de gestion de puissance au niveau transactionnel pour les systèmes sur puce

Nous avons proposé une approche de modélisation structurelle des architectures fonctionnelles augmentées par une stratégie de gestion de puissance au niveau transactionnel. Ensuite, à partir du modèle de structuration de l'architecture, nous avons proposé une méthode pour la génération du code de simulation SystemC-TLM permettant d'évaluer conjointement les comportements fonctionnels et les états induits par cette stratégie. Nous avons respecté un flot de conception standard défini par le formalisme IEEE 1685-2009 IP-XACT. Ce dernier nous ayant permis de décrire les aspects structurels de l'architecture, il ne supporte cependant pas dans sa version actuelle la description des aspects relatifs à la gestion de puissance. C'est pourquoi, nous avons proposé dans un premier temps des extensions au standard afin de pouvoir représenter les différents éléments essentiels pour l'intégration d'une stratégie de gestion de puissance. Dans un deuxième temps, nous avons conçu un générateur de code de simulation SystemC-TLM qui produit la description de la structuration selon la stratégie de gestion de puissance (*power/clock intent*) de la plateforme cible. Ayant ce code généré, nous obtenons une partie importante de la description structurelle du *power/clock intent*. En ajoutant la description du PMU avec ses sous-composants (DPC, CM, ClkST, PST et DST) relatives à la plateforme cible et le scénario d'application exécutée, une simulation de l'architecture complète incluant son comportement fonctionnel et sa gestion de puissance est effectuée.

1.3 Stratégie de gestion de puissance pour les systèmes sur puce à faible consommation basée sur l'analyse des états fonctionnels

Nous avons abordé dans cette contribution la stratégie dynamique de la gestion de puissance. Nous avons proposé une stratégie générique et en ligne basée sur l'observation des états fonctionnels globaux des blocs IP de l'architecture. Avec ces états, nous construisons un graphe de l'historique qui fournit des informations pertinentes permettant de prendre des décisions efficaces de gestion de puissance. En analysant le graphe de l'historique, notre *power management* prend une décision (*power/clock gating*) qui vise à ne pas altérer la performance du système tout en agissant efficacement sur la consommation d'énergie. Néanmoins, le comportement irrégulier des applications implique une variation d'utilisation des ressources matérielles qui peut conduire à des mauvaises prédictions. Afin d'améliorer les décisions de puissance, nous avons ajouté une technique de vieillissement du graphe de l'historique dans le mécanisme de construction de ce graphe de manière à éliminer les états qui pourraient polluer les décisions.

2 Publications de l'auteur liées à cette thèse

- Application and OS unconscious Power Manager for SoC Systems.
  Hend Affes, Amal Chaker, Michel Auguin.
  16th International Symposium on Quality Electronic Design (ISQED),
  Santa Clara, USA, 2015.
- SOC Power Management Strategy Based on Global Hardware Functional State Analysis.
  Hend Affes, Michel Auguin.
  Euromicro Conference on Digital System Design (DSD),
  Funchal, Madeira, Portugal, 2015.
- A Methodology for inserting Clock-Management strategies in Transaction-Level Models of System-on-Chip.
  Hend Affes, Michel Auguin, François Verdier, Alain Pégatoquet.
  Forum on specification & Design Languages (FDL),
  Barcelona, Spain, 2015.
- Clock Management and Analysis for Transaction-Level Virtual Prototypes
  Amal Ben Ameur, Hend Affes, Michel Auguin, François Verdier.
  Forum on specification & Design Languages (FDL),
  Barcelona, Spain, 2015.

– Extending IP-XACT and UPF to support ESL to RTL low power design methodology.
Emmanuel Vaumorin, Grégoire Avot, Hend Affes, Michel Auguin, Alain Pégatoquet, François Verdier.
DAC Workshop on System-to-Silicon Performance Modeling and Analysis,
San Francisco, USA, 2015.

3 Perspectives

Le travail réalisé de cette thèse ouvre plusieurs perspectives pour des travaux futurs. Certaines perspectives seront énumérés dans la suite :

3.1 Application de la stratégie de gestion de puissance basée sur l'observation des états fonctionnels globaux de système sur une architecture augmentée par *clock/power intent*

Nous avons proposé dans cette thèse un moyen de superposer à un prototype virtuel décrit en SystemC-TLM un modèle exécutable représentant un système de gestion de puissance et son contrôle associé (*power/clock intent*) et muni de capacités de vérification. La gestion de puissance considérée se base sur une vue centralisée des états globaux. En effet, une PST et une *ClkST* sont définies afin d'identifier les modes de puissance d'un système. Néanmoins, cette approche est statique et demande une connaissance approfondie des applications exécutées pour expliciter les relations entre les états fonctionnels d'un côté et les contrôles des *Power Domains* et *Clock Domains* de l'autre côté. Afin de rendre cette approche dynamique, un prochain travail est envisageable qui consiste à appliquer la stratégie de gestion de puissance basée sur l'observation des états fonctionnels globaux de système (voir Chapitre V) sur un prototype virtuel augmenté par *clock/power intent*.

3.2 Analyse du comportement thermique d'une architecture augmenté par *clock/power intent*

Les phénomènes thermiques qui interviennent dans les circuits électroniques peuvent être à l'origine d'une dégradation importante de performance du système. Ainsi, l'analyse du comportement thermique dynamique du circuit et le développement de stratégies thermiques adaptées deviennent des considérations primordiales. À partir de l'architecture augmentée par *clock/power*

intent, il est possible d'obtenir la séquence d'activation de chaque *Power Domain* ou *Clock Domain* correspondant à l'exécution abstraite de l'application cible. À ce niveau, une analyse thermique dynamique pourrait être réalisée si un modèle thermique associé à l'architecture est développé. Une trace d'activation des *Power Domains/Clock Domains* par le PMU peut être produite avec la technologie de prototypage virtuel. Cette trace permet de calculer l'évolution dynamique de la température du système et de contribuer à définir une stratégie de gestion thermique. Par ailleurs, une co-simulation entre le modèle thermique et le modèle fonctionnel augmenté par *clock/power intent* peut être réalisée, par exemple en utilisant l'outil *Acceplorer*. Durant cette co-simulation, une stratégie thermique peut être appliquée.

3.3 Étude de l'impact des techniques de réduction de puissance dans les SoC sur la modélisation fonctionnelle SystemC-TLM des blocs IP et de l'architecture

L'exploration des structures du système en *Clock Domains* et en *Power Domains* avec la stratégie de gestion de puissance est nécessaire pour déterminer les bons compromis performance/énergie. Si cette exploration permet de trouver une bonne solution du point de vue de la consommation de l'énergie, il faut s'assurer que cette solution ne remet pas en cause la fonctionnalité souhaitée. Par exemple, éteindre un *Power Domain* peut conduire à des pertes de données dans les composants appartenant à ce domaine, alors que ces données peuvent être nécessaires pour la suite des traitements une fois ces composants remis sous tension. Classiquement, ce type de difficultés est traité d'une part par des registres dits de rétention dans les composants concernés et d'autre part par une gestion de différents signaux de *reset* au niveau de l'architecture qui permettent de définir l'état correct des composants dans ces phases de remises sous tension. En conséquence des solutions efficaces existent à ce jour pour gérer au niveau RTL ces changements d'états dans les composants (par exemple, le standard UPF). Toutefois, les modèles transactionnels au niveau SystemC-TLM ne proposent aucun mécanisme permettant de vérifier si les comportements fonctionnels de ces composants ne sont pas affecté par cette gestion de tension d'alimentation. Par suite, une étude d'intégration de la gestion de signaux de *reset* dans une architecture modélisée en SystemC-TLM dans laquelle est effectuée une gestion de puissance dissipée peut être importante. En effet, quand un composant passe de l'état On à Off, il est nécessaire de modéliser les mécanismes de rétention (sauvegarde) des registres/variables importants dans le composant et lors du passage de l'état Off à On ces registres et variables doivent prendre une valeur particulière (valeurs initiales ou valeurs sauvegardées dans les registres de rétention) en fonction de signaux de reset spécifiques. Il est ainsi

nécessaire de distinguer l'état de mise sous tension initial dans lequel les registres/variables sont tous mis dans un état initial déterminé de la remise sous tension liée au *power management* où la rétention a été appliquée lors du passage à l'état Off.

3.4 Étude d'une approche pour l'optimisation en performance et en énergie d'un système mémoire pour SoC

Étudier comment notre approche (*power/clock intent*) par simulation peut être utilisée comme support à l'analyse de performance et des comportements d'un système mémoire pour un SoC complexe. La principale difficulté réside dans l'identification du compromis précision de simulation versus vitesse de simulation. En effet, pour évaluer les effets sur la performance et l'énergie d'une organisation mémoire ou d'un algorithme intégré dans un contrôleur mémoire il est nécessaire de représenter les événements qui impactent leur états ou qui induisent des prises de décision ayant un effet sur la performance. Par exemple, dans les intervalles de temps où interviennent des interférences entre accès mémoire, ces événements (par exemple, *write-back* et *cache miss* de dernier niveau) doivent être mis en évidence. Ces intervalles de temps où les interférences apparaissent dépendent des scénarios applicatifs considérés, du partitionnement logiciel/matériel, de la politique d'ordonnancement/migration et de la politique de gestion de puissance intégrée dans le SoC. Ainsi, l'ensemble de ces éléments doivent être représentés dans un modèle de simulation avec les mécanismes adaptés pour extraire les informations utiles permettant d'optimiser la performance d'un système mémoire intégré dans un système optimisé en énergie, où la mémoire est justement un des facteurs importants contribuant à la consommation d'énergie. L'idée donc est de concevoir un prototype virtuel enrichi par des mécanismes de gestion de puissance qui peut être utilisé pour produire sur des scénarios réalistes et représentatifs des traces des accès mémoire, et en modélisant ainsi le contrôle relatif à la gestion de la puissance dissipée facilitant l'étude d'une architecture mémoire (hiérarchie et types de contrôleur par exemple).

3.5 Extension de l'aspect de vérification semi formelle orientée consommation d'énergie :

Dans cette thèse, le processus de vérification orienté consommation d'énergie est basé sur l'utilisation d'assertions qui définissent un ensemble de propriétés et de contrats. Au cours de la simulation, ce processus permet la vérification de la cohérence entre le comportement fonctionnel et le comportement résultant de l'application d'une stratégie de gestion d'énergie

à travers le contrôle des contrats prédéfinis. La vérification par assertions présente un point consistant dans notre approche. Toutefois, un ensemble d'améliorations pourraient encore être apportées à ce processus. Nous citons par exemple l'utilisation du standard de spécification des propriétés (PSL) pour une vérification orientée consommation au niveau transactionnel. En effet, la norme IEEE PSL *(Property Specification Language)* [psl] définit des sémantiques puissantes pour les spécifications semi-formelles appliquées à la vérification basée sur des assertions. Dans des travaux récents, certaines couches de ce langage ont été enrichies afin de permettre l'expression d'assertions pour les modèles décrits en SystemC [TKVS08] et SystemC-TLM [FP10] [PFBHA+12]. Cependant, ces efforts n'ont pas encore abordé le problème d'utilisation de PSL pour la vérification des propriétés orientées consommation d'énergie au niveau transactionnel tel que traité dans notre approche. Ainsi, spécifier formellement les exigences orientées consommation d'énergie d'un modèle fonctionnel SystemC-TLM et utiliser ces spécifications avec des moniteurs d'assertions représente un important axe de recherche futur.

Annexe A

Plateforme Audio développée avec *Platform Architect* MCO de *Synopsys*

Pour évaluer notre approche de modélisation d'un *clock intent* dans un outil commercial, nous l'avons appliquée sur un cas d'utilisation développé sous l'outil industriel *Platform Architect* fourni par *Synopsys*.

Cependant, notre approche se base sur une description essentiellement en C++ du clock intent pour mettre en œuvre une approche de modélisation par séparation des préoccupations. En revanche, pour intégrer cette approche dans un flot exclusivement basé SystemC-TLM avec des générations automatiques des fichiers de compilation (*makefiles*) impliquerait d'avoir accès aux générateurs de ces fichiers pour les compléter avec les déclarations de nos modèles et de notre librairie *ClkARCH*. Par conséquent, nous avons élaboré une version modifiée de notre approche en intégrant directement des modèles SystemC-TLM dans la plateforme fonctionnelle. Nous expliquons dans la suite les différentes étapes suivies pour appliquer cette gestion d'horloge sur un prototype virtuel développé sous *Synopsys Virtualizer Platform Creator* contenu dans *Platform Architect MCO*.

A.A Architecture de la plateforme Audio simplifiée augmentée par *clock-intent*

D'abord, nous avons modélisé un prototype virtuel SystemC-TLM sous *Synopsys Virtualizer Platform Creator*. Ce prototype (Figure A.1) représente une version simplifiée de la plateforme Audio décrite dans le chapitre III. Il implémente l'encodage et le décodage G711 et G726 utilisés pour la transmission vocale. Le composant principal du notre prototype est le Cortex M3 ARM qui exécute le logiciel embarqué et stocké dans la mémoire intégrée (MEM). Ce logiciel embarqué implémente l'encodage et de décodage G711 sur des échantillons audio stockés

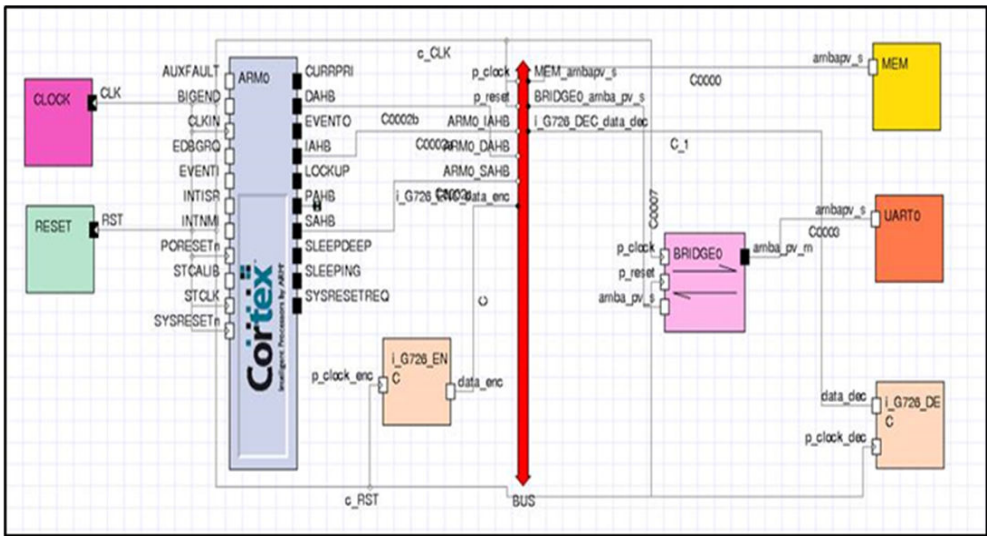


Figure A.1 – Architecture de la plateforme Audio simplifiée

dans la mémoire. Les résultats sont affichés sur la console UART qui est connectée au bus *Advanced Microcontroller Bus Architecture* (AMBA) à l'aide d'un bridge. Le Cortex M3 est le seul initiateur sur le bus AMBA. Ce bus agit comme un routeur surveillant le bon acheminement des transactions TLM-2.0 vers le modèle correct selon le *mapping* mémoire défini. Tous ces blocs IP font partie de la bibliothèque de *Synopsys Virtualizer Platform Creator*. Nous avons ajouté deux blocs SystemC-TLM G726_ENC et G726_DEC connectés sur le bus AMBA comme le montre la Figure A.1. Ces blocs mettent en œuvre respectivement l'encodage et le décodage G726.

Plusieurs scénarios peuvent être exécutés sur cette plateforme. Cette exécution ne nécessite pas qu'une même fréquence de fonctionnement soit appliquée à tous les blocs de l'architecture. Pour gérer les fréquences des différents blocs, nous avons appliqué notre méthodologie de gestion de la distribution d'horloge. Après une analyse du comportement fonctionnel de la plateforme, nous avons décidé de partitionner la plateforme en trois CD comme représenté sur la Figure A.2. Suivant le choix effectué, le Cortex M3 et l'UART sont inclus dans le premier CD nommé «CD_AO». Le deuxième CD *CD_Audio* est composé par les deux blocs G726_ENC etG726_DEC. Un troisième CD inclut l'unité de gestion de puissance (PMU) qui a besoin d'une fréquence stable de fonctionnement pendant toute l'exécution.

Le PMU englobe tous les composants nécessaires pour contrôler la distribution de l'horloge de la plateforme : CM, PM et DPLL. Selon notre approche pour chaque CD, nous associons un CM et un DPLL. Nous avons aussi appliqué une stratégie statique de gestion de puissance mais

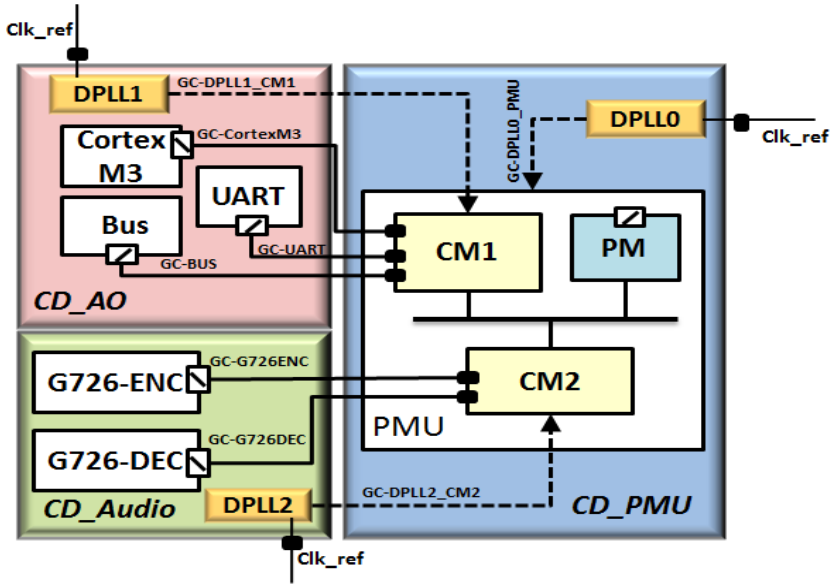


Figure A.2 – Décomposition en clock domain

nous avons défini une version modifiée des tables ClkST et DST afin de faciliter l'interface avec le logiciel exécuté par le Cortex M3. En fait, nous avons défini pour chaque CD une table de mode de puissance contenant les paramètres nécessaires pour basculer d'un OPP à un autre A.3.

State / Ratio	M1	N1	Cortex	Bus	UART
Bypass_mode	1	1	1	1	1
G711_Enc/Dec	5	1	2	5	5
Cortex_Low	5	1	4	5	5
Bus_OFF	5	1	1	off	5
UART_OFF	5	1	1	5	off
CD1_OFF	5	1	off	off	off

(a) Table des OPP du CD : CD_AO

State / Ratio	M2	N2	G726_ENC	G726_DEC
Bypass_mode	1	1	1	1
G726_Enc_Dec	2	1	1	2
G726_ENC_OFF	2	1	off	2
G726_DEC_OFF	2	1	1	off
CD2_OFF	2	1	off	off

(b) Table des OPPs du CD : CD_Audio

Figure A.3 – Tables OPP des *Clock Domains*

Par exemple, la Table A.3(b) associée au «CD_Audio» définit les facteurs de division (M2) et de multiplication (N2) relatifs à la DPLL et les rapports de division relatifs aux blocs inclus dans ce CD et pour les différents modes de puissance. Pour contrôler les blocs d'un CD, le PM envoie des transactions en écrivant des informations de contrôle dans les registres internes du CM correspondant. Les connexions entre les CM et le PM sont réalisées à travers un bus de

type transactionnel TLM2 fourni aussi par la bibliothèque de *Synopsys*. Le PM est un initiateur sur le bus alors qu'il est un esclave sur le bus AMBA fonctionnel.

Pour basculer d'un mode à un autre, le Cortex M3 envoie une requête au PM contenant l'identifiant du CD et un indice de mode de puissance cible. À la réception de cette requête, le PM lit l'identifiant et pointe sur la table de mode de puissance du CD correspondant. Puis, il compare le mode demandé par rapport au mode courant et sélectionne les paramètres qu'il faut changer pour passer au nouveau mode. En effet, deux cas sont possible :(1) Le PM change les facteurs de division et de multiplication relatifs au DPLL (dans ce cas d'utilisation nous n'avons pas implémenté le DPLL_Controller, le changement se fait à travers le PM lui-même) et il envoie les rapports de division au CM correspondant ; (2) Le PM décide de ne pas modifier la fréquence au niveau DPLL et il envoie directement les rapports de division au CM correspondant. Dans la suite, nous présentons le scénario de l'exécution correspond aux modes définis dans la Table A.3(a) et la Table A.3(b) selon la décomposition en CD illustrée sur la Figure A.2. L'exécution de la plateforme démarre par la phase d'initialisation. Le mode *Bypass_mode* est appliqué pour tous les blocs. Vu que nous commençons par l'encodage G711, le Cortex M3 envoie une demande (une écriture dans les registres locaux du bloc PM) au PM pour éteindre le CD *CD_Audio* en basculant ce dernier au mode *CD2_OFF* et une deuxième demande pour appliquer le mode *G711_Enc/Dec* pour le CD : *CD_AO*. Après l'affectation des modes de puissance, le Cortex M3 effectue un codage G711 sur un buffer de k échantillons stockés dans la mémoire de données. Dès que le Cortex M3 termine son traitement et avant d'envoyer les k échantillons codés au bloc G726_ENC, il envoie une nouvelle demande pour activer le G726_ENC (mode *G726_DEC_OFF*) et une autre pour diminuer sa fréquence en demandant la mode Cortex_Low (Table A.3(a)).Le traitement de codage G726 est ensuite effectué par le bloc G726_ENC sur des échantillons stockés dans le buffer d'entrée. Une séquence de contrôle similaire peut être appliquée à la plateforme pour exécuter un scénario de décodage.

En utilisant *Virtuel Analyzer Prototypede Synopsys*, les résultats de l'exécution peuvent être affichés sur la console de l'UART. Un scénario d'encodage/décodage est présenté dans la Figure A.4. Cette figure montre dans un premier temps les échantillons initiaux et ensuite les échantillons encodés par G711 suivis par les échantillons encodés par G726 et enfin les échantillons décodés par G726 suivis par des échantillons décodée par G711.

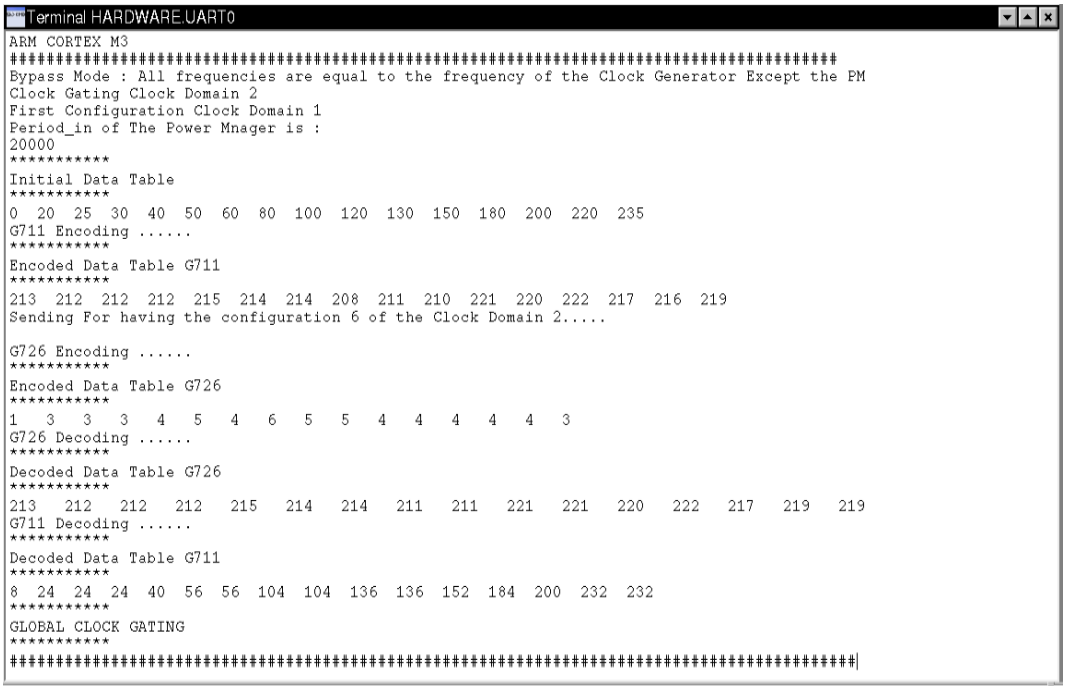


Figure A.4 – Console de l'UART

A.B Évaluation de la consommation d'énergie dynamique

L'outil *VP Explorer* de *Synopsys* permet d'observer via des moniteurs les fréquences des différents composants en utilisant un script TCL sans code intrusif dans ces blocs. En effet, les paramètres à observer sont spécifiés à ces moniteurs à travers ce script TCL qui est appelé dans la console TCL de VP Explorer pendant la simulation de la plateforme. Pour observer la fréquence de chaque bloc, nous définissons un moniteur par bloc.

Par exemple sur le script montré dans la Figure A.7, pour contrôler la fréquence du bloc

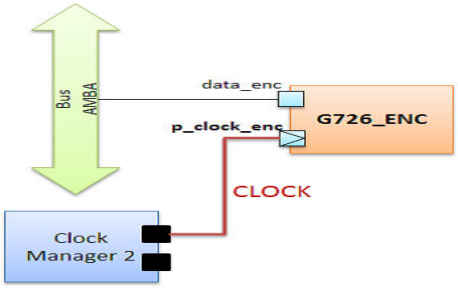


Figure A.5 – Connexions G726 Encoder

G726_ENC, nous avons défini un modèle de puissance appelé «PM0». Puis, nous spécifions que la valeur de la fréquence est récupérée à partir du signal «p_clock_enc» du bloc G726_ENC (voir la Figure A.5).

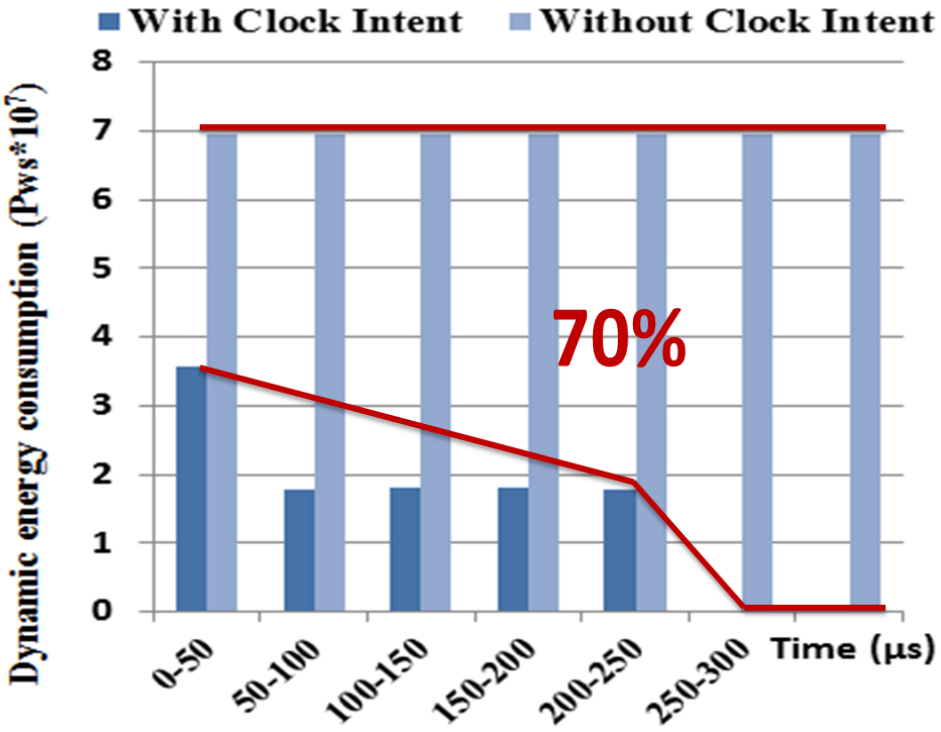


Figure A.6 – Consommation d'énergie de la plateforme sans et avec *clock intent*

Pour évaluer les gains sur la puissance dynamique par notre approche qui opère une modélisation système de l'arbre d'horloge, nous comparons la consommation d'énergie dynamique pour deux cas différents. Dans un premier temps, nous exécutons le scénario encodage/décodage expliqué précédemment sur la plateforme Audio simplifiée sans ajouter le *clock intent*. Ensuite, nous exécutons le même scénario mais sur la plateforme Audio augmentée par le *clock intent* décrit ci-dessus. La consommation totale d'énergie dynamique par intervalle de temps de 50 μs en considérant ces deux cas est illustrée sur la Figure A.6. Comme nous pouvons le remarquer, l'énergie consommée par la plateforme augmentée par le *clock intent* est réduite d'au moins de 70% par rapport l'énergie consommée par la plateforme initiale.

Finalement, les expérimentations effectuées avec l'outil *Platform Architect* montrent que notre méthodologie peut s'intégrer aisément dans des flots de conception industriels basés sur le prototypage virtuel au niveau TLM.

```
##########################################################################################
########

#                    Script TCL for Frequency Monitoring and Power Modeling
#

##########################################################################################
########

#----------------------------------------------------------G726_ENC-----------------------------------
------------------#

# Create the power model with the name pm0 for the clock in of the G726_ENC

global pm0

set pm0 [SNPS_PAM create_monitor {Power Analysis}  PA_0 power_monitor {{{Domain
Name} HARDWARE.i_G726_ENC}{"Aggregation interval" "50 us"}}]

# Define the modem power states (always ends with finalize)

# The dynamic and leakage power consumption per state is defined in Watt

$pm0 add_state i_G726_ENC IDLE   0.005   0.005    100   1.2  1.2

$pm0 add_state i_G726_ENC ACTIVE 0.200   0.005    100   1.2  1.2

$pm0 finalize

$pm0 link_signal_value_to_state /HARDWARE/i_G726_ENC/p_clock_enc
1.9999999999999998E-4 0xf i_G726_ENC IDLE

$pm0 link_signal_value_to_state /HARDWARE/i_G726_ENC/p_clock_enc
199.99999999999997 0xf i_G726_ENC ACTIVE

# set_frequency_input defines how the frequency information is supplied to the power analysis
module

# set_frequency_input input_type

$pm0 set_frequency_input signal

$pm0 set_frequency_signal /HARDWARE/i_G726_ENC/p_clock_enc

# set_voltage_value defines the voltage which is supplied to the power analysis module. default
unit is V
```

Figure A.7 – Script TCL pour la modélisation des paramètres de puissance sous SYNOPSYS

Références bibliographiques

[Ace] *Aceplorer tool : http ://www.doceapower.com/product-services/aceplorer.html*. 44

[AM] Vishwanath Sripath Avinash Mahadeva. *Power Management, Debugging and Optimizations. http ://www.elinux.org/images/2/23/Powermanagement-debugging.pdf*. i, 27

[Ame15] Amal Ben Ameur. Rewriting systemc-tlm functional model to integrate power and clock intent specifications using a meta-modelling approach,. Master's thesis, Université de Nice Sophia Antipolis, 2015. 104

[ASJJM05] Rose Adam, Swan Stuart, Pierce John, and ernandez Jean-Michel. *Transaction Level Modeling in SystemC*. Open SystemC Initiative, 2005. 12

[Ati08] Rabie Ben Atitallah. *Modèles et simulation des systèmes sur puce multiprocesseurs : estimation des performances et de la consommation d'énergie*. PhD thesis, Université des Sciences et Technologies de Lille (USTL), 2008. i, 11

[Ba06] C. Brandolese and al. Dpm at os level : low-power scheduling policies. In *CSECS'06 Proceedings of the 5th WSEAS International Conference on Circuits, Systems, Electronics, Control & Signal Processing*, pages 100–105, 2006. 113

[BANMD07] R. Ben Atitallah, S. Niar, S. Meftali, and J. Dekeyser. An mpsoc performance estimation framework using transaction level modeling. In *Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007. 13th IEEE International Conference on*, pages 525–533, Aug 2007. 36

[BBA11] k. Bhatti, C Belleudy, and M Auguin. Hybrid power management in real time embedded systems : an interplay of dvfs and dpm techniques. *Real-Time Systems*, 2011. 26

[BBDM00] L. Benini, A. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(3) :299–316, June 2000. 22

[BBM04] P. Babighian, L. Benini, and E. Macii. A scalable odc-based algorithm for rtl insertion of gated clocks. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 1, pages 500–505 Vol.1, Feb 2004. 33

[BKS09] F. Bembaron, R. Kakkar, S.and Mukherjee, and A Srivastava. Low power verification methodology using upf. In *Proceedings of DVCon*, pages 228–233, 2009. 35

[BMM13] Tayeb Bouhadiba, Matthieu Moy, and Florence Maraninchi. System-level modeling of energy in tlm for early validation of power and thermal management. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1609–1614, March 2013. 37, 38

[Ca13] S. Cao and al. Energy-efficient stream task scheduling scheme for embedded multimedia applications on multi-issued stream architectures. *Journal of Systems Architecture*, 59(4-5) :187–201, 2013. 113

[Car15] Steve Carlson. Power : What's the problem ? industry trends and solutions in low power design. In *Electronic Design Process Symposium*, Monterey, CA, April 2015. i, 3

[CC07] A. Crone and G. Chidolue. Functional verification of low power designs at rtl. In *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, pages 288–299, 2007. 35

[CG03] L. Cai and D. Gajski. Transaction level modeling : an overview. In *Hardware/Software Codesign and System Synthesis, 2003. First IEEE/ACM/IFIP International Conference on*, pages 19–24, Oct 2003. 12

[CHK14] G. Chen, K. Huang, and A. Knoll. Energy optimization for real-time multiprocessor system-on-chip with optimal dvfs and dpm combination. *ACM Transactions on Embedded Computing Systems (TECS)*, 13, 2014. 113

[Clk] *Shawn McCloud, Calypto Design Systems, Low-Power RTL Report 2012*. i, 23

[cpf11] *S. I. Initiative, Common Power Format (CPF) 2.0 Specification. Silicon Integration Initiative (Si2), Inc., http ://www.si2.org*, 2011. 29

[CSB92] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen. Low-power cmos digital design. *Solid-State Circuits, IEEE Journal of*, 27(4) :473–484, Apr 1992. 24

[dat13] *Data sheets STM32F405xx - STM32F407xx, DocID022152 Rev 4, STMicroelectronics*, June 2013. 115

[DBD+05] Nagu R. Dhanwada, Reinaldo A. Bergamaschi, William W. Dungan, Indira Nair, Paul Gramann, William E. Dougherty, and Ing-Chao Lin. Transaction-level modeling for architectural and power analysis of powerpc and coreconnect-based systems. *Design Autom. for Emb. Sys.*, 10(2-3) :105–125, 2005. 35

[DBN12] S.K. Datta, C. Bonnet, and N. Nikaein. Android power management : Current and future trends. In *Enabling Technologies for Smartphone and Internet of Things (ETSIoT), 2012 First IEEE Workshop on*, pages 48–53, June 2012. 28

[DIBM03] M. Donno, A. Ivaldi, L. Benini, and E. Macii. Clock-tree power optimization based on rtl clock-gating. In *Design Automation Conference, 2003. Proceedings*, pages 622–627, June 2003. 34

[DKV+15] A. Das, A. Kumar, B. Veeravalli, R. Shafik, G. Merrett, and B. Al-Hashimi. Workload uncertainty characterization and adaptive frequency scaling for energy minimization of embedded systems. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, pages 43–48, March 2015. 27

[DOM] *http ://www.jmdoudoux.fr/java/dej/chap-dom.html*. 101

[Don04] A. Donlin. Transaction level modeling : flows and use models. In *Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004. International Conference on*, pages 75–80, Sept 2004. 12

[Duo] *Duolog, site web : http ://www.duolog.com*. 88

[EB00] Frank Emnett and Mark Biegel. Power reduction through rtl clock gating. In *Synopsys Users Group (SNUG)*, 2000. 33

[ecl] *Plugin Eclipse pour l'édition et la vérification des modèles IP-XACT, site web : http ://download.eclipse.org/dsdp/dd/downloads*. 88

[FAGS07] D. Flynn, R. Aitken, A. Gibbons, and K. Shi. *Low Power Methodology Manual*. Springer Publishing Company, Incorporated, 2007. 23, 24

[FCS+01] A.H. Farrahi, Chunhong Chen, A. Srivastava, G. Tellez, and M. Sarrafzadeh. Activity-driven clock design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(6) :705–714, Jun 2001. 33

[FMPP04] F. Fummi, S. Martini, G. Perbellini, and M. Poncino. Native iss-systemc integration for the co-simulation of multi-processor soc. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 1, pages 564–569 Vol.1, Feb 2004. 12

[FP10] L. Ferro and L. Pierre. Formal semantics for psl modeling layer and application to the verification of transactional models. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 1207–1212, March 2010. 139

[Ger00] B. Gerard. Essays in honour of robin milner. In *Proof, Language and Interaction, 2000*, pages 425–454, 2000. 13

[GHF+14] K. Gruttner, P.A. Hartmann, T. Fandrey, K. Hylla, D. Lorenz, S. Stattelmann, B. Sander, O. Bringmann, W. Nebel, and W. Rosenstiel. An esl timing amp ; power estimation and simulation framework for heterogeneous socs. In *Embedded Computer Systems : Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on*, pages 181–190, July 2014. 38

[Goo12] J. Goodacre. The homogeneity of architecture in a heterogeneous world. In *Embedded Computer Systems (SAMOS), 2012 International Conference on*, pages i–i, July 2012. 4

[GSD99] D. Garrett, M. Stan, and A. Dean. Challenges in clockgating for a low power asic methodology. In *Low Power Electronics and Design, 1999. Proceedings. 1999 International Symposium on*, pages 176–181, Aug 1999. 34

[GSGS02] T. Grotker, L. Stan, M. Grant, and S. Stuart. *System Design with SystemC*. Kluwer Academic Publishers, Norwell, MA, USA, 2002. 12, 13

[GY12] D. Greaves and M. Yasin. Tlm power3 : Power estimation methodology for systemc tlm 2.0. In *Specification and Design Languages (FDL), 2012 Forum on*, pages 106–111, Sept 2012. 37

[HH08] N.Z. Haron and S. Hamdioui. Why is cmos scaling coming to an end ? In *Design and Test Workshop, 2008. IDT 2008. 3rd International*, pages 98–103, Dec 2008. 3

[IP-] *The SPIRIT Consortium : IP-XACT http ://www.eda-stds.org/ip-quality-metrics/files/1685-2009.pdf*. i, 6, 18, 19, 87

[JSHP11] H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran. System-level application-aware dynamic power management in adaptive pipelined mpsocs for multimedia. In *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pages 616–623, Nov 2011. 113

[KVG+12] M. Komu, S. Varjonen, A. Gurtov, S. Tarkoma, Geunsik Lim, Changwoo Min, and Youngik Eom. Experiences with power management enabling on the intel medfield phone 35, 2012. 28, 111

[Leb09] H. Lebreton. Modélisation de la consommation électrique en systemc/tlm et réduction de la consommation par le contrôle de méanisme dvfs. Master's thesis, Institut National Polytechnique de Grenoble (INPG), 2009. 36, 38

[LHDX10] Li Luo, Hongjun He, Qiang Dou, and Weixia Xu. Design and verification of multi-clock domain synchronizers. In *Intelligent System Design and Engineering Application (ISDEA), 2010 International Conference on*, volume 1, pages 544–547, Oct 2010. 58

[LJP13] Vincent Legout, Mathieu Jan, and Laurent Pautet. A scheduling algorithm to reduce the static energy consumption of multiprocessor real-time systems. In *Proceedings of the 21st International conference on Real-Time Networks and Systems*, pages 99–108, 2013. 26

[LK09] Bing Li and C.K.-K. Kwok. Automatic formal verification of clock domain crossing signals. In *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*, pages 654–659, Jan 2009. 58

[LM87] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9) :1235–1245, Sept 1987. 123

[LP] *Ultra-low-power 32-bit MCU ARM, http ://www.st.com/web/en/resource/technical/documer* 80

[LV08] H. Lebreton and P. Vivet. Power modeling in systemc at transaction level, application to a dvfs architecture. In *Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual*, pages 463–466, April 2008. 36, 38

[LW13] D. LI and J. WU. Energy-aware scheduling for acyclic synchronous data flows on multiprocessors. *Journal of Interconnection Networks*, (14), 2013. 26

[Mag] *Magillem Design Service, site web : www.magillem.com.* ii, 88, 89

[Mat] *Matthieu Moy. Mini power-aware TLM-platform. http ://www-verimag.imag.fr/m̃oy/ Mini-Power-Aware-TLM-Platform&langf̄r.* 37

[Men] *Mentor Graphics Questa Simulator, http ://www.mentor.com/products/fv/questapower-aware-simulator/.* 35

[Mey92] Bertrand Meyer. Applying "design by contract". *Computer*, 25(10) :40–51, October 1992. 75

[MM13] F. Mischkalla and W. Mueller. Efficient power intent validation using loosely-timed simulation models. In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2013 23rd International Workshop on*, pages 172–179, Sept 2013. 38, 99

[MM14] F. Mischkalla and W. Mueller. Architectural low-power design using transaction-based system modeling and simulation. In *Embedded Computer Systems : Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on*, pages 258–265, July 2014. 38

[MPA11] Ons Mbarek, Alain Pegatoquet, and Michel Auguin. A methodology for power-aware transaction-level models of systems-on-chip using upf standard concepts. In *PATMOS*, pages 226–236, 2011. i, 39, 41, 61, 92

[MPV+13a] T.S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin. Hierarchical power management for asymmetric multi-core in dark silicon era. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pages 1–9, May 2013. 113

[MPV+13b] T.S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin. Hierarchical power management for asymmetric multi-core in dark silicon era. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pages 1–9, May 2013. 121

[MRA10] Amin EL MRABTI. *Méthode et outils de génération de code pour les plateformes multi-cœurs fondés sur la représentation de haut niveau des applications et des architectures*. PhD thesis, Université de Grenoble, 2010. i, 20

[MSKD10] A.K. Mishra, S. Srikantaiah, M. Kandemir, and C.R. Das. Cpm in cmps : Coordinated power management in chip-multiprocessors. In *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, pages 1–12, Nov 2010. 113

[Nau] *Nauet, site web : http ://www.mataitech.com*. 88

[Nel10] Vincent Nelis. *Energy-Aware Real-Time Scheduling in Embedded Multiprocessor Systems,*. PhD thesis, université libre de bruxelles, 2010. 26

[new] *http ://standards.ieee.org/news/2014/ieee_p2415_p2416_wgs.html*. 33

[NLD05] V. Narayanan, Ing-Chao Lin, and N. Dhanwada. A power estimation methodology for systemc transaction level models. In *Hardware/Software Codesign and System Synthesis, 2005. CODES+ISSS '05. Third IEEE/ACM/IFIP International Conference on*, pages 142–147, Sept 2005. 35

[NMM+11] A. Nelson, O. Moreira, A. Molnos, S. Stuijk, B.T. Nguyen, and K. Goossens. Power minimisation for real-time dataflow applications. In *Digital System Design (DSD), 2011 14th Euromicro Conference on*, pages 117–124, Aug 2011. 26

[OMAa] *OMAP35xx Applications Processor, Power, Reset, and Clock Management, Texas Instruments OMAP Family of Products, February 2008, http ://maemo.jacekowski.org/docs/*. i, 24

[OMAb] *Technical Reference Manual, OMAP4470 Multimedia Device Silicon Revision 1.0, Version R, Texas Instruments, Literature Number : SWPU270R, November 2011-Revised April 2013*. 45, 47, 51

[PFBHA+12] L. Pierre, L. Ferro, Z. Bel Hadj Amor, P. Bourgon, and J. Quevremont. Integrating psl properties into systemc transactional modeling #x2014; application to the verification of a modem soc. In *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, pages 220–228, June 2012. 139

[Pow] *Power Compiler, Synpsys Inc. http ://www.synopsys.com.* 33

[psl] *IEEE Standard for Property Specification Language (PSL), 1850-2010.* 139

[RAG02] D. Rainer, G. Andreas, and G.Daniel. *SpecC Language Reference Manual : http ://www.ics.uci.edu/_specc/reference/SpecC LRM 20.pdf.* SpecC Technology Open Consortium, 2002. 16

[Sca] *Scarlet, site web : http ://www.scarletcode.co.uk.* 88

[Seq] *Sequence Design Inc., RTL power estimation, in PowerTheater User guide, release 2007.2, 2007.* 33

[SGM+14] M. Shafique, S. Garg, T. Mitra, S. Parameswaran, and J. Henkel. Dark silicon as a challenge for hardware/software co-design. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2014 International Conference on*, pages 1–10, Oct 2014. 4

[SS11] J. Shinde and S.S. Salankar. Clock gating — a power optimizing technique for vlsi circuits. In *India Conference (INDICON), 2011 Annual IEEE*, pages 1–4, Dec 2011. 34

[ST] *RM0078 Reference manual, SPEAr1340 architecture and functionality, Doc ID 018553 Rev 5, November 2012, STMicroelectronics.* 45, 52

[Syn] *Synopsys tool : http ://www.synopsys.com/Systems/ ArchitectureDesign/Pages/PlatformArchitect.aspx.* 43

[sys] *Open SystemC initiative. SystemC Transactin Level Modeling Library 2.1.0, . http ://www.systemc.org.* i, 11, 16, 18, 73, 91

[Ta11] L. Torres and al. *An Introduction to Multi-Core System on Chip - Trends and Challenges*, chapter Multiprocessor System-on-Chip - Hardware Design and Tool Integration. Springer, 2011. 70

[TKVS08] D. Tabakov, G. Kamhi, M.Y. Vardi, and E. Singerman. A temporal language for systemc. In *Formal Methods in Computer-Aided Design, 2008. FMCAD '08*, pages 1–9, Nov 2008. 139

[TM05] E. Talpes and D. Marculescu. Toward a multiple clock/voltage island design style for power-aware processors. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 13(5) :591–603, May 2005. 58

[TR11] *International Technology Roadmap for Semiconductors, Design*, 2011. i, 3, 5

[upf] *IEEE Standard for Design and Verification of Low Power Integrated Circuits.* i, 29, 30, 61, 65

[VC09] G.B. Vece and M. Conti. Power estimation in embedded systems within a systemc-based design context : The pktool environment. In *Intelligent solutions in Embedded Systems, 2009 Seventh Workshop on*, pages 179–184, June 2009. 42

[Ven] *Vendor Extensions : http ://accellera.org/images/downloads/standards/ip-xact/1685-2009-VendorExtensions-1.0-final.tgz*. 20

[Ver03] *SystemVerilog 3.1 : http ://www.eda.org/sv/SystemVerilog 3.1 final.pdf*, 2003. 16

[Vin13] Lionel Vincent. *Gestion dynamique locale de la variabilité et de la consommation dans les architectures MPSoCs*. PhD thesis, Université de Montpellier 2, 2013. 25

[VPB06] E. Viaud and D. Potop-Butucaru. A new paradigm and associated tools for tlm/t modeling of mpsocs. In *Research in Microelectronics and Electronics 2006, Ph. D.*, pages 217–220, 2006. 10

[WWL03] Li-Chuan Weng, XiaoJun Wang, and Bin Liu. A survey of dynamic power optimization techniques. In *System-on-Chip for Real-Time Applications, 2003. Proceedings. The 3rd IEEE International Workshop on*, pages 48–52, June 2003. 22

[yTS02] J. Bäsig y T. Stöcklein. Handelc : An effective method for designing fpgas and asics. Technical report, GEORGSIMONOHM FACHHOCHSCHULE NÜRNBERG, 2002. 16

[YZ04] Jun Yang and Chuanjun Zhang. Frequent value encoding for low power data buses. *ACM Transactions on Design Automation of Electronic Systems*, 9 :354–384, 2004. 33

[ZAAJ09] H.K. Zrida, M. Abid, A.C. Ammari, and A. Jemai. A yapi system level optimized parallel model of a h.264/avc video encoder. In *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on*, pages 354–361, May 2009. 121

Résumé :

Les systèmes embarqués sur puce (SoC) envahissent notre vie quotidienne. Avec les progrès de la technologie, ils intègrent de plus en plus de fonctionnalités complexes engendrant des besoins accrus en temps de calcul et taille de mémoire. Alors que la complexité de ces systèmes est une tendance clé, la consommation d'énergie est aussi devenue un facteur critique pour la conception de SoC.

Dans ce contexte, nous avons étudié une approche de modélisation au niveau transactionnel qui associe à un modèle fonctionnel SystemC-TLM une description d'une structure de gestion d'un arbre d'horloge décrit au même niveau d'abstraction. Cette structure développée dans une approche de séparation des préoccupations fournit à la fois l'interface pour la gestion de puissance des composants matériels et pour le logiciel applicatif. L'ensemble des modèles développés est rassemblé dans une librairie *ClkARCH*. Pour appliquer à un modèle fonctionnel un modèle d'un arbre d'horloge, nous proposons une méthodologie en trois étapes : la spécification, la modélisation et la simulation. Une étape de vérification en simulation est aussi considérée basée sur des contrats génériques de type assertion.

De plus, nos travaux visent à être compatibles avec des outils de conception actuels. Nous avons proposé une représentation d'une structure de gestion d'horloge et de puissance dans le standard IP-XACT permettant de produire les descriptions C++ des structures de gestion de puissance du SoC.

Par ailleurs, nous avons proposé une approche de gestion de puissance basée sur l'observation globale des états fonctionnels du système dans le but d'éviter ainsi des prises de décisions locales peu efficaces à une optimisation de l'énergie.

Mots Clé : Systèmes sur Puce, Niveau Transactionnel, Gestion et Vérification de l'arbre d'horloge *Clock Domain*, *clock intent*, *power intent*, Assertions, Prédiction de mode de puissance.